Hardware/Software Codesign

0. Organization

doc. dr. Gregor Papa
Overview

- Administration
- Course synopsis
- Introduction and motivation
Organization (1)

- **Lecture:** introductory course + consultations
- **Exercises:** delivered during consultations

- **Contact:** Gregor Papa
gregor.papa@ijs.si
01-477-3514

- **Web page:** http://csd.ijs.si/papa/courses.php
Organization (2)

- **Course materials:**
  - slide copies, exercise sheets, papers
  - the slides contain material from Marco Platzner, Peter Marwedel, Lothar Thiele, Frank Vahid, Reinhard Wilhelm

- **References:**

- **Exam:** written seminar + oral, Slovenian or English
Textbook & slides

- course based
  - on the book and the slides “Embedded System Design” by Peter Marwedel
  - on the slides “Hardware/Software Codesign” by Lothar Thiele
Overview

- Administration
- Course synopsis
- Introduction and motivation
Course Synopsis

- Different Levels of Model Representation
  - Specifications
  - Models
  - Abstraction Levels
- Dealing with Contradictory Constraints
  - Exploration
  - Simulation
    - Worst-Case Execution Time
  - Optimization
- Hardware/Software Mapping
  - Partitioning
  - Scheduling
  - Allocation
- Software Code Optimizations
  - Compilation
- Estimation
Benefits? Learn about …

- … challenges and approaches in modern system design
- … useful optimization methods
- … performance estimation of embedded systems
- … a current research area
Overview

- Administration
- Course synopsis
- *Introduction and motivation*
What is HW/SW Codesign?

- ... integrated design of systems that consist of hardware- and software-components

  - Analysis of HW/SW boundaries and interfaces
  - Evaluation of design alternatives
Hardware/Software Boundaries

- **General purpose systems** (PC, workstation)
  - processor design:
    - processor ↔ compiler, operating system

- **Embedded systems** (cell phone, automotive electronics)
  - design of specialized processors:
    - processor ↔ compiler, operating system
  - system design:
    - processors ↔ dedicated hardware devices
Target Architectures

general-purpose processors

field-programmable gate arrays

microcontrollers

digital signal processors

systems on a chip
Why Codesign? (1)

- Modern embedded systems require "design" optimization
  - many functions, great variability, high flexibility
  - heterogeneous target systems
    - processors, ASICs, FPGAs, systems-on-chip, ...
  - many design goals
    - performance, cost, power consumption, reliability, ...

- Advances in formal / automated design methods
  - automation on the system level becomes possible
  - reduction of cost and time-to-market
Why Codesign? (2)

- Optimization of the “design process”

classic design            co-design

hw                               hw
 sw                               sw
Codesign methodologies

- Different Levels of Model Representation
- Dealing with Contradictory Constraints
- Hardware/Software Mapping
- Software Code Optimizations
- Estimation
System Design

- Specification
- System Synthesis
  - SW-Compilation
  - Instruction Set
  - HW-Synthesis
- Estimation
  - Intellectual Prop. Code
  - Machine Code
- Intellectual Prop. Block
  - Net lists
System Design
Motivation (1)

According to forecasts, future of IT characterized by terms such as
- Disappearing computer,
- Ubiquitous computing,
- Pervasive computing,
- Ambient intelligence,
- Post-PC era,
- Cyber-physical systems.

Basic technologies:
- Embedded Systems
- Communication technologies
“Information technology (IT) is on the verge of another revolution. …..

networked systems of embedded computers ... have the potential to change radically the way people interact with their environment by linking together a range of devices and sensors that will allow information to be collected, shared, and processed in unprecedented ways. ... 

The use … throughout society could well dwarf previous milestones in the information revolution.”

Embedded Systems & Cyber-Physical Systems

“Dortmund“ Definition: [Peter Marwedel]

Information processing systems embedded into a larger product

Berkeley: [Edward A. Lee]:
Embedded software is software integrated with physical* processes. The technical problem is managing time and concurrency in computational systems.

Definition: Cyber-Physical (cy-phy) Systems (CPS) are integrations of computation with physical processes [Edward Lee, 2006].
Ubiquitous computing: Information anytime, anywhere.

Embedded systems provide fundamental technology.

- Communication Technology
  - Optical networking
  - Network management
  - Distributed applications
  - Service provision
  - UMTS, DECT, Hiperlan, ATM

- Embedded Systems
  - Robots
  - Control systems
  - Feature extraction and recognition
  - Sensors/actors
  - A/D-converters

- Pervasive/Ubiquitous computing
  - Distributed systems
  - Embedded web systems
Growing importance of embedded systems

- Spending on GPS units exceeded $100 mln during Thanksgiving week, up 237% from 2006 … More people bought GPS units than bought PCs, NPD found. [www.itfacts.biz, Dec. 6th, 2007]

- …, the market for remote home health monitoring is expected to generate $225 mln revenue in 2011, up from less than $70 mln in 2006, according to Parks Associates. [www.itfacts.biz, Sep. 4th, 2007]

- According to IDC the identity and access management (IAM) market in Australia and New Zealand (ANZ) … is expected to increase at a compound annual growth rate (CAGR) of 13.1% to reach $189.3 mln by 2012 [www.itfacts.biz, July 26th, 2008].

- Accessing the Internet via a mobile device up by 82% in the US, by 49% in Europe, from May 2007 to May 2008 [www.itfacts.biz, July 29th, 2008]
Automotive electronics

- Functions by embedded processing:
  - ABS: Anti-lock braking systems
  - ESP: Electronic stability control
  - Airbags
  - Efficient automatic gearboxes
  - Theft prevention with smart keys
  - Blind-angle alert systems
  - ... etc ...

- Multiple networks
  - Body, engine, telematics, media, safety

- Multiple processors
  - Up to 100
    - 8-bit – door locks, lights, etc.
    - 16-bit – most functions
    - 32-bit – engine control, airbags
  - Processing where the action is
  - Sensors and actuators distributed all over the vehicle
  - Networked together
Avionics

- Flight control systems,
- anti-collision systems,
- pilot information systems,
- power supply system,
- flap control system,
- entertainment system,
- ...

- Dependability is of utmost importance.
Railways

- Safety features contribute significantly to the total value of trains, and dependability is extremely important.
Mobile phones have been one of the fastest growing markets in the recent years,
- Multiprocessor
  - 8-bit/32-bit for UI
  - DSP for signals
  - 32-bit in IR port
  - 32-bit in Bluetooth
- 8-100 MB of memory
- All custom chips
- Power consumption & battery life depends on software

- base stations
  - Massive signal processing
    - Several processing tasks per connected mobile phone
  - Based on DSPs
    - Standard or custom
    - 100s of processors
- Geo-positioning systems,
- Fast Internet connections,
- Closed systems for police, ambulances, rescue staff.
Medical systems

- For example:
  - Artificial eye: several approaches, e.g.:
    - Camera attached to glasses; computer worn at belt; output directly connected to the brain, “pioneering work by William Dobelle”. Previously at [www.dobelle.com]

- Translation into sound; claiming much better resolution.
  [http://www.seeingwithsound.com/etumble.htm]
Extremely Large

- Functions requiring computers:
  - Radar
  - Weapons
  - Damage control
  - Navigation
  - basically everything

- Computers:
  - Large servers
  - 1000s of processors
Inside your PC

- Custom processors
  - Graphics, sound
- 32-bit processors
  - IR, Bluetooth
  - Network, WLAN
  - Harddisk
  - RAID controllers
- 8-bit processors
  - USB
  - Keyboard, mouse
Authentication systems

- Finger print sensors
- Access control
- Airport security systems
- Smartpen®
- Smart cards
- ....
Consumer electronics

- Examples

![Diagram of Ambient Intelligence Global System]

- Ad-Hoc Network of Picocell Ambient Transducers
- 100µW, 1Gops peak (kbps)
- WLAN Basestations
- <1W, 10Gops > 100 Mbps
- www
- Body Transducers
- 100µW - (kbps)
- 10m
- 1000 m
- 10m
- 1m
- SoC-SIP (Wearable) Assistant
  - biometric input
  - global connectivity
  - multimedia, games
  - QoS
  - gps
  - ambient control
  - health...
  - 10..100Gops 0.1-2W
- speak show stimulate
- >100/person aura

© H.De Man/IMEC DATE02
Industrial automation

Examples
Forestry Machines

- Networked computer system
  - Controlling arms & tools
  - Navigating the forest
  - Recording the trees harvested
  - Crucial to efficient work

- Operator panel
  - Graphical display
  - Touch panel
  - Joystick
  - Buttons
  - Keyboard

- “Tough enough to be out in the woods”
Smart buildings

Examples

- Integrated cooling, lightning, room reservation, emergency handling, communication
- Goal: “Zero-energy building”
Robotics

- “Pipe-climber”

- Robot “Johnnie“

- Lego mindstorms
  - Standard controller
    - 8-bit processor
    - 64 kB of memory
  - Electronics to interface to motors and sensors
Estimation

- Hardware, software and system as a whole suitability
Contents

- Levels of Abstraction in Electronic System Design
- Typical Design Flow of Hardware-Software Systems
The reason for buying is not information processing.
a an ist i t a t o ms
Cell Processor B combines:

- A general-purpose architecture core with
- Coprocessing elements which greatly accelerate multimedia and vector processing applications, as well as many other forms of dedicated computation.
omm ni atin Em st ms

- sensor networks in civil engineering, buildings, environmental monitoring, traffic, emergency situations
- smart products, wearable ubiquitous computing
Large-scale Distributed Systems

New Applications and System Paradigms
om a ison

- **Embedded Systems**
  - New applications that are known at design-time
  - Not programmable by end user
  - Additional computing power not useful
  - Criteria
    - Cost
    - Power consumption
    - Predictability
    - Meeting time bounds

- **General Purpose Computing**
  - Broad class of applications
  - Programmable by end user
  - Faster is better
  - Criteria
    - Cost
    - Average speed
Increasing application complexity even in standard and large volume products
- Large systems with legacy functions
- Mixture of event-driven and data flow tasks
- Examples multimedia, automotive, mobile communication

Increasing target system complexity
- Mixture of different technologies, processor types, and design styles
- Large systems-on-a-chip combining components from different sources, distributed system implementations

Numerous constraints and design objectives
- Examples cost, power consumption, timing constraints, dependability
- Dynamic environments
- Capture the required behavior
- Validate specifications
- Efficient translation of specifications into implementations
- How can we check that we meet real-time constraints
- How do we validate embedded real-time software? Large volumes of data, testing may be safety-critical.
n a i it

- ES must be
  - probability of system working correctly provided that it was working at $t$
  - probability of system working correctly $d$ time units after error occurred
  - probability of system working at time $t$
  - no harm to be caused
  - confidential and authentic communication

- Even perfectly designed systems can fail if the assumptions about the workload and possible errors turn out to be wrong. Making the system dependable must not be an after-thought; it must be considered from the very beginning.
ES must be efficient
  - ode-si e efficient especially for systems on a chip
  - Run-time efficient
  - eight efficient
  - ost efficient
  - Energy efficient
any ES must meet
- real-time system must react to stimuli from the controlled object or the operator within the time interval by the environment
- or real-time systems, right answers arriving too late are wrong
- all other time-constraints are called
- guaranteed system response has to be explained without statistical arguments
a  im  st  ms

- Embedded and Real-time
  Synonymous
  - ost embedded systems are real-time
  - ost real-time systems are embedded
a t i     i s st ms

- Typically, ES are
  - Behavior depends on input
    - Automata model appropriate,
      model of computable functions inappropriate

- i s st ms
  Analog digital parts
towards a certain knowledge about behavior at design time can be used to minimize resources and to maximize robustness.

- no mouse, keyboard and screen
Contents

- What is an Embedded System

- Typical design flow of Hardware-Software Systems
Stationary analysis

- Formal description of selected properties of a system or subsystem
- Model consists of data and associated methods

- Degree of abstraction, granularity
  - System, architecture, logic, transistor,
  - Module, block, function,
- View
  - Behavior, structural, physical

- Linking adjacent levels of abstraction refinement
- Stepwise adding of structural information
s o s t a t i o n s

Process module

Structure

Behavior
Contents

- What is an Embedded System
- Levels of Abstraction in Electronic System Design
Sinaoas

- **Simulation** is the process of generating the description of a system in terms of related lower-level components from some high-level description of the expected behavior.

The “describe-and-synthesize” paradigm by a ski, 4

In contrast to the traditional “specify-explore-refine” approach, also known as “design-and-simulate” approach, manual design steps are more error-prone than automatic synthesis and, therefore, simulation is more important.
Software synthesis and code generation

- hardware synthesis
- interface and communication synthesis
- hardware-software partitioning and component selection
- hardware-software scheduling

- application specification
- design space exploration and system optimization
- estimation
Relating the problem level with the implementation level
Partitioning of system function to programmable components software, hard-wired or parameterized components hardware or application specific instruction set processors.

- to scheduling and load distribution problem in real-time operating systems
  - time constraints, context switch and context switch overhead, process synchronization and communication
  - to real-time operating systems
    - larger design space with very different solutions
    - high optimization requirements motivation for hardware design
    - underlying hardware is not fixed
Similarity to allocation or load distribution problem in high-level synthesis or real-time operating systems
The principle of synthesis based on abstraction only makes sense if there are available:

- Estimate properties of the next layer(s) of abstraction
- Design decisions are based on these estimated properties

If the estimation is not correct or not accurate enough, the design will be sub-optimal or even not working correctly.
System Design

- Specification
- System Synthesis
- Estimation
- SW-Compilation
- Instruction Set
- HW-Synthesis
- Intellectual Prop. Code
- Machine Code
- Intellectual Prop. Block
- Net lists
Consider a simple example:

The observer pattern defines a one-to-many dependency between a subject object and any number of observer objects so that when the subject object changes state, all its observer objects are notified and updated automatically.

Example server pattern in action:

```java
public oid addListener(listener)

public oid setValue(newvalue)

my value = newvalue

for int i = 0; i < mylisteners.length; i++
    myListeners[i].valueChanged(value)
```

Will this work in a multithreaded context?
server pattern it m te es

public synchronized oid add instender listener

public synchronized oid setValue newValue

my alue ne alue

for int i i mylisteners.len th i
my isteners i . alueChan ed ne alue

a asoft recommends a ainst this.

What s ron ith it
**Te es sing monitors are mine ie ds**

```java
public synchronized void addListener(Listener listener)
```

```java
public synchronized void setValue(newValue)
```

```java
for (int i = 0; i < myListeners.length; i++)
    myListeners[i].value Changed(newValue)
```

`valueChanged` may attempt to acquire a lock on some other object and stall. If the holder of that lock calls `addListener` deadlock

```
mute
```

```
calls add listener
```

```
lock
```

```
re leases
```

```
held y
```

---

2-
Simple observer pattern gets complicated

```java
public synchronized void addListener(Listener listener)

public void setValue(newValue)

synchronized this
    myValue = newValue
    listeners = myListeners.clone

for (int i = 0; i < listeners.length; i++)
    listeners[i].valueChanged(newValue)
```

While holding lock, make a copy of listeners to avoid race conditions.

Notify each listener outside of the synchronized lock to avoid deadlock.

His still isn't right.
What's wrong with it?
Simple observer pattern

to make it right

```java
public synchronized void addListener(listener)

public void setValue(newValue)
    synchronized this
    my value new value
    listeners my listeners.clone

for int i = listeners.length;
    listeners[i].valueChanged(newValue)
```

Suppose two threads call setValue. One of them will set the value last. It's possible that value in the object will change in the opposite order, but listeners may be notified in the opposite order. The listeners may be alerted to the value-changes in the reverse order.
Proems intoreadasedconcurreny

- Nontrivial software written with threads, semaphores, and mutexes is incomprehensible to humans.

- Search for non-thread-based models which are the requirements for appropriate specification techniques.
StateCharts

ata- lo Models
Humans not capable to understand systems containing more than a few objects.

Most actual systems require more objects:

- Hierarchy
  - Examples states processes procedures.
  - Examples processors racks printed circuit boards
- Requirements or Specifications

- Components send streams of data to each other.

- Overstressed or
What does it mean to compute

Ode s o comp tation de inition

- Components and an execution model for computations for each component
- Communication model for exchange of information between components.
  - Shared memory
  - Message passing
Shared memory

- Potential race conditions inconsistent results possible
- Critical sections sections at which exclusive access to resource is shared memory must be guaranteed.

```
process a
.. P S  obtain lock
   .. critical section
   S  release lock

process
.. P S  obtain lock
   .. critical section
   S  release lock
```

ace-free access to shared memory protected by S possible

his model may be supported by
- mutual exclusion for critical sections
- cache coherency protocols
on occurrence async message passing

- Sender does not have to wait until message has arrived
  potential problem: buffer overflow

send

receive
oc ing sync rono s message passing

- Sender will wait until receiver has received message
Sync rono s message passing  

- SP communication between processes
  Hoare
  *rendez-vous*-ased communication
  Example

```plaintext
process
  ..
  ar a ...
  a
  c a -- output
end

process B
  ..
  ar ...
  ...
  c -- input
end
```
components

- on Neumann model
  
  Sequential execution program memory etc.

- discrete event model

```
a queue
  a
  c
```

- time action

  ```
a  c  a  a
  ```
Components

- Finite state machines

- Differential equations

\[ \frac{\partial^2 x}{\partial t^2} = b \]
D hard-ware description language is commonly used as a design-entry language for digital circuits.
Sensitivity lists are a shorthand for a single `always`-statement at the end of the process body.

```vhdl
process y
  begin
    prod and y
  end process
end process
```
No language that meets all language requirements can be used without compromises.
Contents

- Models of Computation
- Data-IO Models
Classical automata

input $X$  \rightarrow \text{Internal state } Z  \rightarrow \text{output } Y

clock

Next state $Z$ computed $\delta$ function

Output computed $\lambda$ function

Moore-automata

$Y \lambda \ Z \ Z \ \delta \ X, \ Z$

Mealy-automata

$Y \lambda \ X \ Z \ Z \ \delta \ X, \ Z$
State arts

Classical automata not useful for complex systems. Complex graphs cannot be understood by humans.

StateCharts Harel
Introducing hierarchy

SM will be in exactly one of the substates of S if S is active either in or in B or ..
Definitions

- Current states of SMs are also called states.
- States which are not composed of other states are called .
- States containing other states are called .
- or each basic state s the super-states containing s are called .
- Super-states S are called if exactly one of the sub-states of S is active hence S is active.
Dea t State ecanism

ry to hide internal structure from outside orld
efault state
illed circle indicates su-state entered here er super-state is entered. Not a state y itself
History and default mechanisms can be used hierarchically.

or input m. S enters the state it as in before S as left can be B, C, or E. If S is entered for the very first time, the default mechanism applies.

History and default mechanisms can be used hierarchically.
omining istory and Dea t State

same meaning
Convenient ways of describing concurrency are required.

- FSM is in all (immediate) sub-states of a super-state.
Entering and Leaving AND-Super-States

Line-monitoring and key-monitoring are entered and left, when service switch is operated.
ree representation state sets
Computation of state sets from leaves to root:

- Basic states: state set state
- Super-states: state set union of children
- Super-states: state set artesian product of children

\[
\begin{align*}
Q_H &= Q_L, 
Q_G &= Q_I \cup Q_K \\
Q_F &= Q_G \times Q_H, 
Q_B &= Q_C \cup Q_D \\
Q_E &= Q_F \cup Q_M, 
Q_A &= Q_B \times Q_E \\
Q_A &= (Q_C \cup Q_D) \times (Q_M \cup ((Q_I \cup Q_K) \times Q_L))
\end{align*}
\]
In Statecharts, states are either

- - - - - -
Since time needs to be modeled in embedded systems, timers need to be modeled. In Statecharts, special edges can be used for timeouts. If event a does not happen while the system is in the left state for 20 ms, a timeout will take place.
singers in Answering a line

Lproc

4 s

lift off

talk

return (callee)

dead

timeout

play text

beep

8 s record

timeout

beep

silent
epresentations

esides states, arbitrary many other variables can be defined. This way, not all states of the system are modeled explicitly.

These variables can be changed as a result of a state transition. State transitions can be dependent on these variables.

![Diagram with labeled parts: action, condition, variables, unstructured state space]
General Edge Labels

- event
- condition
- action

- List only for the next evaluation of the model and can be either internally or externally generated.

- Refer to values of variables that keep their value until they are reassigned.

- An either be assignments for variables or creation of events.

- service-off not in Lproc service:
Events and actions can be composed of several events:

- **and** 2: event that corresponds to the simultaneous occurrence of e and e.
- **or** 2: event that corresponds to the occurrence of either e or e or both.
- **not** : event that corresponds to the absence of event e.

- can also be composed:
  - **and** 2: actions a and a are executed in parallel.

- All events, states and actions are globally visible.
ow are edge labels evaluated

:  
  . ffect of e TERNAL changes on events and conditions is evaluated,
  . he set of transitions to be made in the current step and right
  hand sides of assignments are computed,
  . ransitions become effective, variables obtain new values.
Example

In a single phase environment, executing the left state first would assign the old value of b (\( b \)) to a and b. Executing the right state first would assign the old value of a (\( a \)) to a and b. The execution would be non-deterministic.
execution of a Statehart model consists of a sequence of (status, step) pairs

Status values of all variables, set of events, current time
Step execution of the three phases

Status

phase

phase
In an actual clocked (synchronous) hardware system, both registers would be swapped as well.

Same separation into phases found in other languages as well, especially those that are intended to model hardware.
nfortunately, there are several time-semantics of State Harts in use. This is another possibility:

- step is executed in arbitrarily small time.
- Internal (generated) events exist only within the next step.
- Internal events can only be detected after a stable state has been reached.
Examples

State diagram:

---

stable state

---

state diagram:
Example

- on-determinism

State diagram:

\[ A, B \xrightarrow{a} C, D \]

\[ a \rightarrow E, H \]

\[ a \rightarrow F, G \]
E a ple

state diagram (only stable states are represented, only a and b are eternal):

\[
\begin{align*}
\text{a} & \quad \vee \quad \text{a} \\
\text{a} & \quad \downarrow \\
\text{a} & \quad \downarrow \\
\text{a} & \quad \vee \quad \text{a}
\end{align*}
\]
Evaluation of Statearts

- Allows arbitrary nesting of - and -super states.
- In a follow-up paper to original paper.
- Large number of commercial simulation (StateMate, StateFlow Matlab, etterState, ML, ...)
- Available back-ends translate Statearts into , thus enabling software or hardware implementations.
Evaluating State Arts

- generated, not useful for applications,
- o description of - ,
- o - ,
- o description of .
SDL

- (S L) is a specification language targeted at the unambiguous specification and description of the behaviour of reactive and distributed systems.

- sed here as a (prominent) e ample of a model of computation based on as n r n us essage passing.

- appropriate also for distributed systems
Unification of SDL's

Communication between FSMs (or processes) is based on message-passing, assuming a potentially indefinite large queue.

- Each process fetches the next entry from the queue,
- checks if input enables transition,
- if yes: transition takes place,
- if no: input is discarded (exception: S-mechanism).
Deter inisti

- Let tokens be arriving at FF at the same time:
  - order in which they are stored, is unknown

Il orders are legal: simulators can show different behaviors for the same input, all of which are correct.
Contents

- Models of computation
  - Statecharts
  - -
Data I  Language  del

- communicating through

```
  process
     F F  buffer
  process
     F F  buffer
  process
     F F  buffer
```

2-
il s p Data I Languages

- imperative language style: program counter is king
- dataflow language: movement of data is the priority
- Scheduling responsibility of the system, not the programmer

- All processes run simultaneously
- Processes can be described with imperative code
- Processes can communicate through buffers
- Sequence of read tokens is identical to the sequence of written tokens
Data I   Languages

- Appropriate for applications that deal with:
  - Fundamentally concurrent: maps easily to parallel hardware
  - Perfect fit for block-diagram specifications (control systems, signal processing)
  - Matches well current and future trend towards multimedia applications

- :
  - ost Language (process description), e.g. , , ava, ..... .
  -oordination Language (network description), usually home made, e.g. ML.
An MPEG-2 video decoder application structured as a Kahn Process Network.
a n r e s s Net r s

- Proposed by Ahn in as a general-purpose scheme for parallel programming:
  - destructive and blocking (reading an empty channel blocks until data is available)
  - non-blocking
  - infinite size

- Unique attribute:
From Ahn's original paper

process f(in int u, in int v, out int w)

int i  bool b  true
for ( )
i  b  wait(u) : wait(v)
printf(in, i)
send(i, w)
b  b

hat does this do

Process alternately reads from u and v, prints the data value, and writes it to w
From Ansh's original paper:

process g(in int u, out int v, out int w)

int i bool b  true
for()
i wait(u)
if (b) send(i, v) else send(i, w)
b  b

What does this do?

Process reads from u and alternately copies it to v and w.
From ahn's original paper:

process h(in int u, out int v, int init)

int i init
send(i, v)
for( )
i wait(u)
send(i, v)

hat does this do
process sends initial value, then passes through values.
What does this do?

Prints an alternating sequence of $s$ and $g$.

Sends a $h$ once and then copies input to output.

$m$ sends a $h$ once and then copies input to output.
Determina

- System is random if the information about the system and its inputs is not sufficient to determine its outputs.

- Define the history of a channel to be the sequence of tokens that have been both written and read. Process network is said to be error free if the histories of all channels depend only on the histories of the input channels.

- Functional behavior is independent of timing (scheduling, communication time, execution time of processes).

- Separation of functional properties and timing.
Determinacy

monotonic mapping

\[ [x_1, x_2, x_3, \ldots] \rightarrow F \rightarrow [y_1, y_2, y_3, \ldots] \]
Determinacy

- **orma de inition**

  $\{x_1, x_2, x_3, \ldots\} \xrightarrow{} F \xrightarrow{} \{y_1, y_2, y_3, \ldots\}$

  - $[x_1, x_2, x_3, \ldots]$
  - $[x_1] \subseteq [x_1, x_2] \subseteq [x_1, x_2, x_3, \ldots]$
  - $\forall 1 \in \subseteq F$
Determinism

- determinate
  - Reasoning
    - , y, y
      - y
    - y, , y
    - y
  - , y, y
    - y
  - , y, y
in n eterminacy

- y
- ...
- ...
- ...

- amp e

2 - 6
in neterminacy

1  [ ]
F    [ , ]
F
2  [ ]

1  [ , ]
F    [ , , ]
F
2  [ ]

\[ \subseteq [ ], [ ] \subseteq [ , ], [ ] \]
\[ F \not\subseteq F \]
\[ [ , ] \not\subseteq [ , , ] \]
o nded memor

- tart ith o nded er si es
- any s hed in te hni e
- itho t dead o y ontin e
- y dead o, in rease si e
From finite to finite er i e

- $y$
- $n$
- $n$
- $y$

2

$2^{-}$
Deacame

x

2

y

1, 1, 1,
am e Finite i e er in
- action
- x
- y
- on
- x   y
- y   y
- x   y
Data DF

- ync

- n

- Data

- DF

- estri
tion

- i
ed
m
er
o
token

- amp

e

1
<table>
<thead>
<tr>
<th>DF c e in</th>
</tr>
</thead>
<tbody>
<tr>
<td>- c ed e  y at compi e time</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>-         y</td>
</tr>
<tr>
<td>y</td>
</tr>
<tr>
<td>sta ish re ati e e e tion rates y  y</td>
</tr>
<tr>
<td>eetermine eriod e hed e y  y</td>
</tr>
<tr>
<td>- Ret t x  y</td>
</tr>
</tbody>
</table>
\[
\begin{bmatrix}
3 & -2 & 0 & 0 \\
0 & 4 & 0 & -3 \\
0 & 1 & -3 & 0 \\
-1 & 0 & 2 & 0 \\
-2 & 0 & 0 & 1 \\
\end{bmatrix}
\begin{bmatrix}
a \\
b \\
c \\
d \\
\end{bmatrix} = 0
\]
in the a ancin ati n

- ain D c ed ing t eorem

- \[ \begin{bmatrix} n \\ y \\ x \\ n1 \end{bmatrix} \]

- \[ \begin{bmatrix} n1 \\ x \end{bmatrix} \]

- \[ \begin{bmatrix} y \\ y \end{bmatrix} \]

- \[ y \]

- **amp e**

\[
\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 4 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 & -2 & 0 & 0 \\ 0 & 4 & 0 & -3 \\ 0 & 1 & -3 & 0 \\ -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = 0
\]
Determine the incidence matrix for the graph: 

\[
\begin{bmatrix}
  a \\
  b \\
  c \\
  d \\
\end{bmatrix}
= 
\begin{bmatrix}
  2 \\
  3 \\
  1 \\
  4 \\
\end{bmatrix}
\]

...
temp De i n
Example 1: Evolutionary Algorithms for DSE

- selection
- recombination
- mutation

"chromosome" = encoded allocation + binding

1. Allocation
2. Binding
3. Decode allocation
4. Decode binding
5. Scheduling
6. Fitness evaluation
7. Design point (implementation)
8. Fitness
9. User constraints

1. Selection
2. Recombination
3. Mutation
Definition: A specification graph is a graph $G_S=(V_S,E_S)$ consisting of a data flow graph $G_P$, an architecture graph $G_A$, and edges $E_M$. In particular, $V_S=V_P \cup V_A$, $E_S=E_P \cup E_A \cup E_M$.
Example 1: Mapping
Example 1: hallenges

- ncoding of (allocation+binding)
  - simple encoding
    - e.g., one bit per resource, one variable per binding
    - easy to implement
    - many infeasible partitioning solutions
  - encoding + repair
    - e.g., 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 partitioning solutions
- eparation of the initial population
- ecombination
Example 1: ase Study

behavioral specification of a video codec for video compression
Example 1: ase Study

problem graph of the video coder
am le o t are t es s

ABABABCCABABA

1 S

2 C

CODE(A)
CODE(B)
CODE(A)
CODE(B)
CODE(C)

CALL(A)
CALL(B)
CALL(A)
CALL(B)
CALL(C)

FOR 1 TO 2
CODE(A)
CALL(B)
CODE(C)
CODE(A)
PROCEDURE A
FOR 1 TO 3
CALL(A)
CODE(B)
CODE(B)
3-2
RISC 1  DSP 1  BUS 1  NoC  BUS 2  RISC 2  DSP 2

Tile 1  --------  Tile 2

am le ard are rc tecture
am le esult o u ct o al mulat o

\[ n(p) \]

<table>
<thead>
<tr>
<th>number of activations</th>
<th>1</th>
<th>2</th>
<th>297000</th>
<th>891000</th>
<th>297000</th>
<th>594000</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>transferred data (in MByte)</td>
<td>b(s)</td>
<td>15.0</td>
<td>519.4</td>
<td>421.4</td>
<td>421.4</td>
<td>975.6</td>
<td>228.2</td>
</tr>
</tbody>
</table>

\[ p \]

\[ s \]
<table>
<thead>
<tr>
<th>Process</th>
<th>Runtime on RISC</th>
<th>Runtime on DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>dispatch gop</td>
<td>0.13</td>
<td>0.20</td>
</tr>
<tr>
<td>dispatch macroblock</td>
<td>6.68</td>
<td>8.52</td>
</tr>
<tr>
<td>dispatch block</td>
<td>0.06</td>
<td>0.04</td>
</tr>
<tr>
<td>transform block</td>
<td>2.00</td>
<td>1.25</td>
</tr>
<tr>
<td>collect block</td>
<td>0.05</td>
<td>0.04</td>
</tr>
<tr>
<td>collect macroblock</td>
<td>12.33</td>
<td>8.51</td>
</tr>
<tr>
<td>collect gop</td>
<td>0.18</td>
<td>0.30</td>
</tr>
</tbody>
</table>
\[
obj_1 = \max_{c \in \mathcal{C}} \left\{ \sum_{\forall p \text{ mapped to } c} n(p) \cdot r(p, c) \right\}
\]

\[
obj_2 = \max_{g \in \mathcal{G}} \left\{ \sum_{\forall s \text{ mapped onto } g} \frac{b(s)}{t(g)} \right\}
\]
ard are/ o t are odes

stem mulat o

doc dr re or a a
utl e

- D S
- S C
- S H A
stem a d odel

- A

  I S D

- T

- A

- system

- input

- output

- model g
\[ x(t) = f(x(t), u(t), t) \]
\[ x(t_0) = x_0 \]
\[ u(t) \]
\[ y(t) = g(x(t), u(t), t) \]
\[ u_c = -\frac{1}{RC} u_c + \frac{1}{RC} u \]

\[ y = u - u_c \]
me

- l

- s

- p
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
screte  e  t  stems

- A D S

- A  
- D S

- T  
- D S
- I  
- D S

- O

- P
medre s etdre

- T
  - T
- A

- 11
me dr e s e t dr e

- S
- A

- 12
screte  e  t  odel   a  d   mulat o
om o e ts o a screte e t mulat o
Discrete event simulation

- I
- D
- U

- t route
- le
- set to
- process b
  call subs stem
  module s
  remo e
  e e t
  rom
- u date stat st cal
  ormat o
- e erate s mulat o
  re ort
Problem: Within the same simulation cycle, “cause” and “effect” events share the same time of occurrence.

Solution: The simulator uses a zero duration virtual time interval, called *delta-cycle* ($\delta$).

- The role of a *delta-cycle* is to order “simultaneous” events within a simulation cycle, i.e. identifying which event caused another; “causes” and “effects” are separated by *delta-cycles*.

Simulation cycles may be composed of several *delta-cycles* ($\delta$).
Outline

- System Classification
- Discrete Event Simulation
- Example SystemC
- Simulation at High Abstraction Levels
SystemC in a nutshell

- A “system-level” modeling language
  - Several levels of abstraction (from purely functional to cycle accurate pin-accurate)
  - Special attention to systems with embedded software
- A library of C++ templates and classes for modeling concurrent systems
  - Hardware-oriented data types
  - Communication mechanism
  - Concurrency model
- An event-driven simulation kernel for executing models
- Available for free (Windows + Linux)


```cpp
#include "systemc.h"

SC_MODULE(nand) { // declare a NAND sc_module
    sc_in<bool> A, B; // input signal ports
    sc_out<bool> F; // output signal ports

    void do_it() { // a C++ function
        F.write( !(A.read() && B.read()) );
    }
}

SCCTOR(nand) { // constructor for the module
    SCMETHOD(do_it); // register do_it() w/ kernel
    sensitive << A << B; // sensitivity list
}
};
```
```c
#include "nand.h"
SC_MODULE(exor) {
    sc_in<bool> A, B;
    sc_out<bool> F;
    nand n1, n2, n3, n4;
    sc_signal<bool> S1, S2, S3;

    SC_CTOR(exor) : n1("N1"), n2("N2"), n3("N3"), n4("N4") {
        n1.A(A);
        n1.B(B);
        n1.F(S1);

        n2 <<= A << S1 << S2;

        n3(S1);
        n3(B);
        n3(S3);

        n4 <<= S2 << S3 << F;
    }
};
```
Modules: the building blocks of SystemC models
Processes: basic units of functionality

- **SC_THREADS**
  - Can be suspended (wait())
  - Implicitly keep state of execution

- **SC_METHODs**
  - Execute their body from beginning to end
  - Simulate faster
  - Do not keep implicit state of execution

- Processes must be contained in a module
Module

SC_MODULE(Module_name) {
    // Declare ports, internal data, etc.
    // Declare and/or define module functions

    SC_CTOR(Module_name) {
        // Body of the constructor

        // Process declarations and sensitivities
        SC_METHOD(function1);
        sensitive << input1 << input2;

        SC_THREAD(function2);
        sensitive << input1 << clk;
    }
};
Processes can directly communicate through signals.
SystemC introduces general purpose primitives

- **Channel**
  A container for communication and synchronization, e.g. can have state and private data, transport data, transport events.
  They implement one or more *te aces*

- **ter ace**
  Specify a set of access methods to the channel
  But it does not implement those methods

- **E e t**
  le ible, low level synchronization primitive, used to construct other forms of synchronization
  Have no type and no value

- **ther comm. sync. models can be built based on the above primitives**
Wait and Notify

- **Wait**: halt process execution until event is raised
  - `wait()` with arguments => dynamic sensitivity
    - `wait(sc_event)`
    - `wait(time)`
    - `wait(time_out, sc_event)`

- **Notify**: raise an event
  - `notify()` with arguments => delayed notification
    - `my_event.notify();` // notify immediately
    - `my_event.notify(SC_ZERO_TIME);` // notify next delta cycle
    - `my_event.notify(time);` // notify after `time`
initialization Phase

date signals

Complete the set of ready processes

Number of ready processes

date signals

execute all the processes until alignment

execute all the processes until alignment

delta signals
Channel

Producer -> Write Interface -> Read Interface -> Consumer

FIFO
class write_if : public sc_interface
{
    public:
        virtual void write(char) = 0;
        virtual void reset() = 0;
};

class read_if : public sc_interface
{
    public:
        virtual void read(char&) = 0;
        virtual int num_available() = 0;
};
class fifo: public sc_channel,
   public write_if,
   public read_if
{
    private:
     enum e {max_elements=10};
     char data[max_elements];
     int num_elements, first;
     sc_event write_event,
     read_event;
    bool fifo_empty() {...};
    bool fifo_full() {...};

    public:
    fifo() : num_elements(0),
     first(0);

    void write(char c) {
     if (fifo_full())
      wait(read_event);
     data[<you calculate>] = c;
     ++num_elements;
     write_event.notify();
    }

    void read(char &c) {
     if (fifo_empty())
      wait(write_event);
     c = data[first];
     --num_elements;
     first = ...
     read_event.notify();
    }

    void reset() {
     num_elements = first = 0;
    }

    int num_available() {
     return num_elements;
    }
}; // end of class declarations
SC_MODULE(producer) {
    public:
        sc_port<write_if> out;
    SC_CTOR(producer) {
        SC_THREAD(main);
    }
    void main() {
        char c;
        while (true) {
            out.write(c);
            if (...) out.reset();
        }
    }
};

SC_MODULE(consumer) {
    public:
        sc_port<read_if> in;
    SC_CTOR(consumer) {
        SC_THREAD(main);
    }
    void main() {
        char c;
        while (true) {
            in.read(c);
            cout<<
            in.num_available();
        }
    }
};

SC_MODULE(top) {
    public:
        fifo afifo;
        producer *pproducer;
        consumer *pconsumer;
    SC_CTOR(top) {
        pproducer=new producer("Producer");
        pproducer->out(afifo);
        pconsumer=new consumer("Consumer");
        pconsumer->in(afifo);
    };
}
constgen.h

SC_MODULE(constgen) {
    sc_fifo_out<float> output;

    // The constructor
    SC_CTOR(constgen) {
        SC_THREAD(generating);
    }

    void generating() {
        while (true) {
            output.write(0.7);
        }
    }
}
adder.h

SC_MODULE(add) {
    sc_fifo_in<float> input1, input2;
    sc_fifo_out<float> output;
    SCCTOR(add) {
        SC_THREAD(adding());
    }
    void adding() {
        while (true) {
            output.write(input1.read() + input2.read());
        }
    }
}
forker.h

```c
SC_MODULE(forker) {
    sc_fifo_in<float> input;
    sc_fifo_out<float> output1, output2;

    SCCTOR(forker) {
        SC_THREAD(forking());
    }

    void forking() {
        while (true) {
            float value = input.read();
            output1.write(value);
            output2.write(value);
        }
    }
}
```

The will deadlock unless an initial token is put into the loop:

```c
output1.write(0.0);
```
```c
#include <stdio.h>

SC_MODULE(printer) {
    sc_fio_in<float> input;

    SC_CTOR(printer) {
        SC_THREAD(printing);
    }

    void printing() {
        for (unsigned int i = 0; i < 100; i++) {
            float value = input.read();
            printf("%f\n", value);
        }
        return; // this indirectly stops the simulation
        // (no data will be flowing any more)
    }
}
```
```c
#include "constgen.h"
#include "adder.h"
#include "forker.h"
#include "printer.h"

int sc_main(int argc, char* argv[]) {

    // The FIFO channels
    sc_fifo<float> gen_add, add_fork, fork_add, fork_print;

    // The modules
    constgen Gena("Generator");
    adder Addy("Adder");
    forker Forky("Forker");
    printer Prn("Printer");

    Prn.input(fork_print);
    sc_start(); // run forever
    return 0;
}
```
SystemC can emulate virtually any discrete time system
- RTL hardware modeling
- Static/Dynamic multirate dataflow
- Kahn process network
- Transaction-based

Multiple different MoCs in the same system
- Natural way of expressing complex systems
- MoC can “evolve” during refinement process
- Communication/synchronization between different MoCs via channels (adapters)
tline

- System lassification
- iscrete vent imulation
- am le ystem
- i atio at i t a tio
Functional

- nti ed n tional e el
  - se: model un timed functionality
  - ommunication: shared variables messages
  - y ical languages: atla

Transaction level

- ransa tion e el
  - se: o architecture analysis early development timing estimation
  - ommunication: method calls to channels
  - y ical languages: ystem

Register transfer level

- e ister ransfer e el Pin e el
  - se: design and verification
  - ommunication: wires and registers
  - y ical languages: erilog
**Abstraction Models**

- Time granularity for communication computation objects can be classified into 3 basic categories: **Timed**, **Approximate-Timed**, **Cycle-Timed**.
- Models B, C, D, and E could be classified as **Transaction Level Models (TLM)**.

**A.** "Un-timed functional model"

**B.** "Timed functional model"

**C.** "Transaction model"

**D.** "Cycle-accurate communication model"

**E.** "Cycle-accurate computation model"

**F.** "Register transfer model"

---

**System Modeling Graph**
(2003 Dan Gajski and Lukai Cai)
A "Un-Timed Functional Model"

- Computation
  - Un-timed behavior
- Communication
  - Un-timed transfer
  - Variables

Diagram:
- Un-timed execution: B, B2, B3
- Parallel execution: B2, B3
- Sequential execution: B2, B3

- Computation
  - Sequential execution: B2, B3
  - Parallel execution: B2, B3

- Variables: B2, B3

Diagram:
- Un-timed execution:
  - B: v, a, a
  - B: v2, v, b, b
  - B: v3, v, b, b

- Parallel execution:
  - B2, B3

- Variables: B2, B3

- Functions:
  - v, v2, v3
  - u(v)
B Timed Functional Model"

- Computation (on processing elements - Es)
  - Time annotation (estimate)

- Communication
  - Message passing no protocol implementation
  - Un-timed transfer

- Mappin
  - Es (architecture) allocation and process-to-E mapping

![Diagram showing computation and communication with nodes labeled A, B, C, D, and E, with arrows indicating flow and time annotations.]
Compile generated C and run natively

Analyse basic blocks compute delays

delay characterisation

Annotate C code

Model C code execution delay

Compile generated C and run natively

Example B Software Code Annotation

**Specification**

**A C nput**

**Analyze basic blocks compute delays**

**Delay characterisation**

**Annotate C code**

**Model C code execution delay**

**Compile generated C and run natively**

v__st_tmp = v__st;
startup(proc);
if(events[proc][0] & 1)
execute(proc);

**UT A C source code**

**UT UT functionally equivalent C code augmented by execution times**

v__st_tmp = v__st;
DELAY(LI+LI+LI+LI+LI+LI+OPc);
startup(proc);
if(events[proc][0] & 1) {
  DELAY(OPi+LD+LI+OPc+LD+OPi+OPi+IF);
  execute(proc);
}
C: “Transaction Model”

- Computation
  - Approximate-timed (estimate)
- Communication
  - Approximate-timed (estimate) using simplified (abstract) bus protocols
- Mapping
  - Mapping of computation and communication

```
v1 = a*a;
v2 = v1 + b*b;
v3 = v1 - b*b;
v4 = v2 + v3;
c = sequ(v4);
```

Diagram:
- PE1: B1, v1 = a*a;
- PE2: B2, v2 = v1 + b*b;
- PE3: B3, v3 = v1 - b*b;
- PE4 (Arbiter): cv11, cv2, cv12

Nodes:
- Master interface
- Slave interface
- Arbiter interface
D: “C ircle Accurate Communication Model”

- Computation
  - Approximate-timed (estimate)

- Communication
  - Protocol bus channels (time cycle-accurate and pin-accurate)

- Mapping
  - Mapping of computation and communication

---

**Computation**
- $v1 = a^2a$
- $v2 = v1 + b^2b$
- $v3 = v1 - b^2b$
- $v4 = v2 + v3$
- $c = \text{sequ}(v4)$

**Communication**
- Protocol bus channels
  - Master interface
  - Slave interface
  - Arbiter interface

**Mapping**
- Mapping of computation and communication

---

**Diagram**
- PE1
- PE2
- PE3
- PE4 (Arbiter)
- B1
- B2
- B3
- B4
- Address 1
data 31
ready
ack
- 1 Master interface
- 2 Slave interface
- 3 Arbiter interface

---

**Notes**
- Approximate-timed
- Cycle-timed
- Un-timed

---

E: "C icle Accurate Communication Model"
E: "Cycle Accurate Computation Model"

- **Computation**
  - Cycle-accurate

- **Communication**
  - Approximate-timed (estimate) using simplified (abstract) bus protocols

- **Wrappers**
  - Simulation interfaces between cycle-accurate PEs and abstract bus channels interfaces
Example E: IS at is an ISS

- An Instruction Set Simulator (ISS) is a coded in a which mimics the behavior of a processor by “reading” instructions and maintaining internal variables which represent processor’s registers.

- Instruction-accurate
- Cycle-accurate

- Simulate (execute and monitor) machine code instructions, compiled for a target processor.
Example E: Types of ISS

**Interpretive ISS**

```
int Reg[32];
...
while(1) {
    Fetch();
    Decode();
    Execute();
    InterruptHandler();
}
```

```
switch INSN {
    case ADD: r3=r1+r2;
    case SUB: ...
}
```

**Compiled ISS**

```
... add r1, r2, r3 ...
... add(r1, r2, r3);
...
```

```
#define Add(r1, r2, r3)\r3=r1+r2
```

original C code

compilation

original assembly code
eister Transfer odel

- Computation and Communication
  - cycle timed
  - modeled on the level of combinatorial (stateless) functions, memory and digital signals

E1, E2: microprocessors
E3, E: custom hardware
### Different Abstraction Models

<table>
<thead>
<tr>
<th>Models</th>
<th>Communication time</th>
<th>Computation time</th>
<th>Communication Scheme</th>
<th>E Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Unimed Functional model</td>
<td>o</td>
<td>o</td>
<td>variables</td>
<td>no E</td>
</tr>
<tr>
<td>B. Unimed Functional model</td>
<td>o</td>
<td>approximate</td>
<td>bstract channel</td>
<td>bstract</td>
</tr>
<tr>
<td>C. Transaction model</td>
<td>approximate</td>
<td>approximate</td>
<td>bstract channel</td>
<td>bstract</td>
</tr>
<tr>
<td>D. Cycle Accurate Communication model</td>
<td>Cycle acc rate</td>
<td>approximate</td>
<td>rotocol b s channel</td>
<td>bstract</td>
</tr>
<tr>
<td>E. Cycle Accurate Computation model</td>
<td>approximate</td>
<td>Cycle acc rate</td>
<td>bstract b s channel</td>
<td>in acc rate</td>
</tr>
<tr>
<td>F. Register Transfer model</td>
<td>Cycle acc rate</td>
<td>Cycle acc rate</td>
<td>wires</td>
<td>in acc rate</td>
</tr>
</tbody>
</table>
Trace ased Sim lation

- (Un timed Functional odel) and (ransaction odel)
  - Higher simulation speed (for large hardware software systems, multiprocessors)
  - Uses estimates of non functional beha ior
Traceased Simulation: 2 ases

- **Input:** application specification
- **Output:** execution traces = sequence of events \( \in \{ ; ; \} \)
- **Method:** untimed functional simulation

- **Input:**
  - execution traces
  - architecture specification
  - mapping specification
- **Output:** performance estimation results, e.g. execution time, processor load and bus load
- **Method:** map abstract read, write and compute primitives onto virtual machines that reflect binding and resource sharing (mapping)

race generation

race based simulation
Cosimulation motivation for fixed models and the simulation is very much dependent on the system description model.

How to see several abstraction levels or several models of computation:

- Different abstraction levels
- Different description languages
- Different models of computation

<table>
<thead>
<tr>
<th>more abstract</th>
<th>less abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>packet</td>
<td>address</td>
</tr>
<tr>
<td></td>
<td>data</td>
</tr>
<tr>
<td></td>
<td>cmd</td>
</tr>
<tr>
<td></td>
<td>cnfg</td>
</tr>
<tr>
<td></td>
<td>status</td>
</tr>
</tbody>
</table>

Discrete | Continuous
Cosimulation Example

Environments for multiprocessor system cosimulation:

- Several ISSs coupled with H R simulation: accurate, but slow (especially for multiple ISS running in parallel)

- ISSs are replaced with higher level simulation models: speed up simulation time
ard are Soft are Codesign

orst Case Execution Time Analysis

doc dr re or apa
System Specification

- Specification
- System Synthesis
- Estimation
- Instruction Set
- Synthesis
- Compiler
- Instruction Set
- Net lists
- Instruction Set
- Synthesis
- Compiler
- Instruction Set
- Net lists

Intellectual Code

- Intellectual
- Intellectual
- Intellectual
- Intellectual
- Intellectual
- Intellectual

Machine Code

- Machine Code
- Machine Code
- Machine Code
- Machine Code
- Machine Code
- Machine Code
Contents

- problem statement, tool architecture
- rogram ath Analysis
- alue Analysis
- Caches
  - must, may analysis
- ipelines
  - Abstract pipeline models
  - Integrated analyses
Industrial Needs

- , often in safety critical applications abound
  - Aeronautics, automotive, train industries, manufacturing control

Sideairbag in car,
Reaction in 1 mSec

Free stream air velocity

- ingibration of airplane,
sensing every mSec
**Embedded Time Systems**

- Embedded controllers are expected to finish their tasks reliably within time bounds.
- As scheduling must be performed.
- Essential: of all tasks statically known.
- Commonly called the - (CE)
- Analogously, (BCE)
**Industry's best practice**

or s if either worst case input can be determined, or exhaustive measurement is performed otherwise, determine upper bound from execution times of instructions
Measurements: determine execution times directly by observing the execution or a simulation on a set of inputs.
  - Does not guarantee an upper bound to all executions.

Exhaustive execution in general not possible!
  - Too large space of input domain x set of initial execution states.

Compute upper bounds along the structure of the program:
  - Programs are hierarchically structured.
  - Statements are nested inside statements.
  - So, compute the upper bound for a statement from the upper bounds of its constituents.
Sequence of Statements

Constituents of A:
A1 and A2

Upper bound for A is the sum of the upper bounds for A1 and A2

\[ ub(A) = ub(A1) + ub(A2) \]
on t ona Statement

Constituents of A:

\[ \text{Condition} \]

\[ \text{Statements} \ A_1 \text{ and } A_2 \]

\[ \text{ub}(A) = \text{ub}(\_ ) + \max(\text{ub}(A_1), \text{ub}(A_2)) \]
\[ A \equiv \text{for } i \leftarrow 1 \text{ to } 1 \text{ do } A1 \]

\[
\begin{align*}
\text{ub}(A) &= \text{ub}(i \leftarrow 1) + \\
&\quad 1 \times (\text{ub}(i \leq 1) + \\
&\quad \text{ub}(A1)) + \\
&\quad \text{ub}(i \leq 100)
\end{align*}
\]
to start

assignment

\[ x \leftarrow a + b \]

\[ \text{ub}(x \leftarrow a + b) = \]
\[ \text{cycles}(\text{oa } a) + \]
\[ \text{cycles}(\text{oa } ) + \]
\[ \text{cycles}(a ) + \]
\[ \text{cycles}(\text{store } e ) \]

\[ \text{store } x \]

\[ \text{cycles} \]

\[ \text{ot } a \quad i \quad a \quad e \]
\[ \text{to } \quad \text{ode } n \quad o \quad \text{esso } s \]

\[ \text{move } 1 \]
Modern processors *increase performance* by using:

- **Case 1:** everything goes smoothly: no cache miss, operands ready, needed resources free, branch correctly predicted.
- **Case 2:** everything goes wrong: all loads miss the cache, resources needed are occupied, operands are not ready.

These features make **CE computation difficult**: execution times of instructions vary widely.

- **Best case**
  - everything goes smoothly: no cache miss, operands ready, needed resources free, branch correctly predicted.
- **Worst case**
  - everything goes wrong: all loads miss the cache, resources needed are occupied, operands are not ready.
\[ x = a + b; \]

```
LOAD    r2, _a
LOAD    r1, _b
ADD     r3, r2, r1
```
mmccents an enates

- _iming ccident_ cause for an increase of the execution time of an instruction
- _iming enalt_ the associated increase
- _pes_ of timing accidents
  - ache misses
  - Pipeline stalls
  - ranch mispredictions
  - us collisions
  - emory refresh of D A
  - T miss
Micro-architecture analysis:
- Uses Abstract interpretation
- Excludes as many Timing Accidents as possible
- Determines $T$ for basic blocks (in contexts)

Cost-Optimal Timing Determination
- Maps control flow graph to an integer linear program
- Determines upper bound and associated path
Contents

- Introduction
  - problem statement, tool architecture
- ROGRAM ATH NALYSIS
- Value Analysis
- Aches
  - must, may analysis
- Pipelines
  - Abstract pipeline models
  - Integrated analyses
what_is_this {  
  read (a,b);  
  done = FALSE;  
  repeat {  
    if (a>b)  
      a = a-b;  
    elseif (b>a)  
      b = b-a;  
    else done = TRUE;  
  } until done;  
  write (a);  
}
Problem: The number of possible program paths grows exponentially with the program length.

Model:
- Fixed number of cycles for each basic block (from static analysis).
- Loops must be bounded.

Concept:
- Transform structure of the program into a set of (integer) linear equations.
- Solution of the integer linear program (P) yields bounds on the T.
**as c oc**

- **e initiation** A basic block is a sequence of instructions where the control flow enters at the beginning and exits at the end, without stopping in between or branching (except at the end).

\[
\begin{align*}
  \text{t1} & := c - d \\
  \text{t2} & := e \times \text{t1} \\
  \text{t3} & := b \times \text{t1} \\
  \text{t4} & := \text{t2} + \text{t3} \\
  \text{if t4} & < 10 \text{ goto L}
\end{align*}
\]
**as c** oc s

1. **etermine basic bloc s o a program**: 
   - the first instruction targets of unconditional umps
   - instructions that follow unconditional umps

2. **dete ine t e asi o s**: 
   - there is a basic bloc for each block beginning
   - the basic bloc consists of the block beginning and runs until the next block beginning (exclusive) or until the program ends
generated control flow graph $C$
- the nodes are the basic blocks

```
i := 0
t2 := 0
L
  t2 := t2 + i
  i := i + 1
  if i < 10 goto L
x := t2
```
/ * k >= 0 */

s = k;
WHILE (k < 10) {
    IF (ok)
        j++;
    ELSE {
        j = 0;
        ok = true;
    }
    k ++;
}

r = j;
**Definition:** A program consists of $N$ basic blocks, where each basic block $B_i$ has a worst-case execution time $c_i$ and is executed for exactly $x_i$ times. Then, the WCET is given by

$$WCET = \sum_{i=1}^{N} c_i \cdot x_i$$

- the $c_i$ values are determined using the static analysis.
- how to determine $x_i$?
  - structural constraints given by the program structure
  - additional constraints provided by the programmer (bounds for loop counters, etc.; based on knowledge of the program context)
Structural Constraints

Flow equations:

\begin{align*}
    &d_1 = d_2 = x_1 \\
    &d_2 + d_8 = d_3 + d_9 = x_2 \\
    &d_3 = d_4 + d_5 = x_3 \\
    &d_4 = d_6 = x_4 \\
    &d_5 = d_7 = x_5 \\
    &d_6 + d_7 = d_8 = x_6
\end{align*}
Itional Constraints

\begin{align*}
s &= k; \\
\text{WHILE} \ (k < 10) \\
\text{if} \ (ok) \\
\quad j &= j + 1; \\
\quad j &= 0; \quad \text{ok} = \text{true}; \\
k &= k + 1; \\
r &= j;
\end{align*}

loop is executed for at most 10 times

\[ x_3 = 10 \ x_1 \]

B5 is executed for at most one time

\[ x_5 = 1 \ x_1 \]
ILP with structural and additional constraints:

\[
\text{program is executed once}
\]

\[
= \max \left\{ \sum_{i=1}^{N} i \cdot i \right\} \quad 1 = 1 \wedge
\]

\[
\sum_{j \in \text{in}(B_i)} j = \sum_{k \in \text{out}(B_i)} k = i, i = 1 \ldots N \wedge
\]

additional constraints

structural constraints
Contents

- Introduction
  - problem statement tool architecture
- program at nal sis
- alu nal sis
- access
  - must ma anal sis
- ipelines
  - strict pipeline models
  - integrated anal ses
Abstract Interpretation AI

- *abstract analysis* or static program analysis

- *abstract algebra* errororm the program's computations using value descriptions or *abstract values* in place of the concrete values start it a description of all possible inputs

- supports *correctness proofs*
Abstract Intervals

- **Abstract domain** related to concrete domain
  - Abstract and concrete functions
  - E.g. \( \text{Intervals} \rightarrow \text{Intervals} \), where \( \text{Intervals} = \text{LB} \times \text{UB} \), \( \text{LB} = \text{UB} = \text{Int} \cap \{\infty, -\infty\} \) instead of \( L \rightarrow \text{Int} \)

- **Abstract transfer functions** for each statement type – abstract versions of their semantics
  - E.g. \( + : \text{Intervals} \times \text{Intervals} \rightarrow \text{Intervals} \) where \([a,b] + [c,d] = [a+c, b+d] \) with + extended to \(-\infty, \infty\)

- **A join function** combining abstract values from different control-flow paths
  - E.g. \( t : \text{Interval} \times \text{Interval} \rightarrow \text{Interval} \) where \([a,b] t [c,d] = [\min(a,c), \max(b,d)]\)
Value Analysis

**Motivation:**
- Provide access information to data-cache/pipeline analysis
- Detect infeasible paths
- Derive loop bounds

**Method:** calculate intervals at all program points, i.e. lower and upper bounds for the set of possible values occurring in the machine program (addresses, register contents, local and global variables).
Value Analysis

Intervals are computed along the edges. Edges are unioned.

- Move #4, D0
  - D: [-, +], [x, x]

- Add D1, D0
  - D: [+, ], [-, ], [x, x]

- Move (A0, D0), D1
  - Access [x, x]

Which address is accessed here?
Introduction
- problem statement, tool architecture

Program Path analysis

value analysis

aches
- must, may analysis

Pipelines
- abstract pipeline models
- Integrated analyses
aches are used, because

- ast main memory is too expensive
- he speed gap between PU and memory is too large and increasing

aches wor well in the average case:

- Programs access data locally (many hits)
- Programs reuse items (instructions, data)
- ccess patterns are distributed evenly across the cache
Processor

Cache

Memory

access takes ~ 1 cycle

access takes ~ 100 cycles

fast, small, expensive

(relatively) slow, large, cheap
PU wants to read write at the address $a$, sends a request for $a$ to the bus.

- **Cases:**
  - Block $m$ containing $a$ in the cache (hit): request for $a$ is served in the next cycle.
  - Block $m$ not in the cache (miss): $m$ is transferred from main memory to the cache, $m$ may replace some block in the cache, request for $a$ is served asap while transfer still continues.

- Several replace-ent strategies: LRU, PLRU, I,... determine which line to replace.
ach cache set has its own replacement policy = cache sets are independent. Everything explained in terms of one set

- 

  - e ace ent trate :

    - replace the block that has been Least Recently Used
    - updated by ages

- a a e: -way set associative cache

<table>
<thead>
<tr>
<th>access</th>
<th>age</th>
<th>age</th>
<th>age</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>m (miss)</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>m (hit)</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
<tr>
<td>m (miss)</td>
<td>m</td>
<td>m</td>
<td>m</td>
<td>m</td>
</tr>
</tbody>
</table>
analytic Analysis

- How to statically precompute cache contents:

  - Must nas is:
    - or each program point (and calling context), find out which blocks are in the cache.
    - Determines safe information about cache hits. Each predicted cache hit reduces  
  
  - Ma nas is:
    - or each program point (and calling context), find out which blocks may be in the cache. Complement says what is not in the cache.
    - Determines safe information about cache misses. Each predicted cache miss increases B  

5 - 3
Cache contents depends on the context, i.e. calls and loops

- First iteration loads the cache:
  - Intersection loses most of the information.

- Distinguish as many contexts as useful:
  - Unrolling for caches
  - Unrolling for branch prediction (pipeline)
Introduction
  - problem statement, tool architecture

Program Path analysis

value analysis

aches
  - must, may analysis

In e ines
  - abstract pipeline models
  - Integrated analyses
single cycle

multiple cycle

pipelining
Ideal Case: 1 Instruction per Cycle
Several instructions can be executed in parallel.

Some pipelines can begin more than one instruction per cycle: VLIW, Superscalar.

Some CPUs can execute instructions out of order.

: Hazards and cache misses.
Pipeline hazards:
- : operands not yet available
  at dependencies
- : Consecutive instructions use same resource
- : Conditional branch
- : Instruction fetch causes cache miss
Program execution order (in instructions)

- 40 beq $1, $3,
- 44 and $12, $2, $5
- 48 or $13, $6, $2
- 52 add $14, $2, $2
- 72 lw $4, 50($7)

Time (in clock cycles)

CC 1  CC 2  CC 3  CC 4  CC 5  CC 6  CC 7  CC 8  CC 9

Diagram of instruction flow through different stages (IM, DM, Reg).
Program execution order (in instructions):

- sub $2, $1, $3
- and $12, $2, $5
- or $13, $6, $2
- add $14, $2, $2
- sw $15, 100($2)

Time (in clock cycles):

<table>
<thead>
<tr>
<th>Value of register $2</th>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>10/-20</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
<td>-20</td>
</tr>
</tbody>
</table>
: prediction of cache hits on instruction or operand fetch or store

l z r4 2 r1

: analysis of data control hazards

add r4 r5 r6
l z r7 1 r1
add r8 r4 r4

: analysis of resource hazards
Processor pipeline cache memory inputs viewed as performing transitions every clock cycle.

Starting in an initial state or an instruction transitions are performed until a state is reached:
- $f_1$: instruction has left the pipeline
- $f_2$: execution time of instruction

- $u$: interprets instruction stream starting in state $s$ producing trace
- $c$: successor basic block is interpreted starting in initial state $las$
- $1$: gives number of cycles
t|ct|e|e|o|B|c|B|oc

- $u\text{cto}e\text{ec}:c\text{oc}s:t\text{ct}e\text{e}t\text{te}:t\text{ce}$
  - interprets instruction stream $o$ annotated with cache information starting in state $s$ producing trace
  - $le\ h$ gives number of cycles

- Abstract states may lack information e.g. about cache contents.
- Assume local worst cases is same in the case of no timing anomalies
- Traces may be longer but never shorter.
or successor basic block. In particular, if there are several predecessor blocks:

- sets of states
- combine by assuming that local worst case is same
Using statically computed effective addresses and loop bounds

- Assume cache hits here predicted
- Assume cache misses here predicted or not excluded.
- Only the first result states of an instruction need to be considered as input states or successor instructions.
Deceotectuemo
to
coto
to
chtectue

to
to
E
to

multi objective optimization
- Optimization
- Design
- Implementation
What are randomized search heuristics

- applicable to black box optimization problems

- by iteratively improving a solution by variation and selection
  - can find many different optimal solutions in a single run
- choose subset that maximizes overall profit
- minimizes total weight
there is no single optimal solution, but some solutions are better than others when selecting a solution. Inding the good solutions.
o che

- pro it more important than cost ranking
- eight must not exceed 24 g constraint
selects one solution considering constraints

decision making often easier

evolut. algorithms well suited

searches or a set of green solutions

searches or one green solution

ranks objectives against constraints

Whe to e the Dec o
Hence, classical single-echelon methods

- simulated annealing
- tabu search
- integer linear program
- other constructive or iterative heuristic methods

- evolutionary algorithms
- genetic algorithms

- is done after the optimization process.

- is done after the optimization process.

- is done after the optimization process.
parameter oriented scaling dependent

set oriented scaling independent
Weighed Opt Functions

- Multiple objectives: $y_1, y_2, y_k$
- Parameters
- Transformation
- Single objective: $y$

Example: Eighthing approach

- $y_1, y_2, y_k$

Minimization problem
ttot Ecodoutto
population

archive

sample
evary
select

update
truncate

population

archive
E out o o th ct o

min. \( y_1 \)

max. \( y_2 \)

hypothetical trade o ront
B c Box t o

Objective vector

decision vector

e.g. simulation model

Optimization Algorithm:
only allowed to evaluate direct search
Design 

Specification → Optimization → Evaluation → Implementation

[Diagrams and images related to the process steps]
Embedded Internet Devices

Mobile Internet

Wearable Computing

Core
et o oce o

et o oce o high performance programmable device designed to efficiently execute communication
orkloads r le e al
incoming los packet streams routing or arding transcoding outgoing los processed packets encryption decryption
real time los

e.g. voice

e.g. stp
non real time los

- 2
<table>
<thead>
<tr>
<th>Task Specification</th>
<th>Specification of the task structure to encode each task to the corresponding tasks to be executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usage Scenarios</td>
<td>Different usage scenarios to encode sets of loads with different characteristics</td>
</tr>
<tr>
<td>Network Processor Implementation</td>
<td>Architecture, task mapping, and scheduling</td>
</tr>
</tbody>
</table>
| Objective          | 1. Maximize performance  
                      | 2. Minimize cost  
                      | 3. Memory constraint  
                      | 4. Delay constraints |
Ex ot o tt te

or each usage scenario separately

allocation bindings

performance cost vector

architecture template

task graph

binding restrictions

architecture

performance

for each usage scenario separately
ectue e o

- Introduction
- Design
- Implementation
design point is dominated by if:
- better or equal than in all criteria and
- better in at least one criterion.

point is Pareto optimal or a are points if it is not dominated.

The domination relation imposes a partial order on all design points
- We are faced with a set of optimal solutions.
- Convergence of solutions vs. convergence.
Definition 1 (Dominance relation)
Let \( f, g \in \mathbb{R}^m \). Then \( f \) is said to dominate \( g \), denoted as \( f \succ g \), iff

1. \( \forall i \in \{1, \ldots, m\} : f_i \geq g_i \)

2. \( \exists j \in \{1, \ldots, m\} : f_j > g_j \)

Definition 2 (Pareto set)
Let \( F \subseteq \mathbb{R}^m \) be a set of vectors. Then the Pareto set \( F^* \subseteq F \) is defined as follows: \( F^* \) contains all vectors \( g \in F \) which are not dominated by any vector \( f \in F \), i.e.

\[
F^* := \{ g \in F | \forall f \in F : f \succ g \} \tag{1}
\]
Maximize $y_1, y_2, \ldots, y_k \quad f_1, f_2, \ldots, f_n$

- Pareto optimal: not dominated
- Incomparable
- Worse

Set of all Pareto optimal solutions
Randomized search algorithm

do ed B c Box e ch o th

de  
indo good solutions without investigating all solutions  
uto better solutions can be found in the neighborhood of good solutions  
inormation available only by unction evaluations

t randomly choose a solution \( t_1 \) to start with  
\( f \)  
\( t \geq t \) randomly choose a solution \( t_1 \) using solutions \( 1 \rightarrow t \)
E ≥ 1
- evolutionary algorithm

1
- tabu search

1
- simulated annealing

- both

- no mating selection

- no mating selection
Limitations of Randomized Search Algorithms

The No-Free-Lunch Theorem

All search algorithms provide in average the same performance on all possible functions with finite search and objective spaces.

[Wolpert, McReady: 1997]

Remarks:
- Not all functions equally likely and realistic
- We cannot expect to design the algorithm beating all others
- Ongoing research: which algorithm suited for which class of problem?
course Synopsis

- Introduction
- Optimization
- Implementation
Design choices

- representation
- fitness assignment
- mating selection
- parameters
- environmental selection
- variation operators
Comparison of Three Implementations

- Objective knapsack problem

Tradeoff between distance and diversity?
esign hoices

fitness assignment  mating selection

parameters

environmental selection  variation operators
Representation

<table>
<thead>
<tr>
<th>search space</th>
<th>decoder</th>
<th>solution space</th>
<th>objectives</th>
<th>objective space</th>
</tr>
</thead>
</table>

- solutions encoded by vectors, matrices, trees, lists, ...

**issues:**
- completeness  each solution has an encoding
- uniformity  all solutions are represented equally often
- redundancy  cardinality of search space vs. solution space
- feasibility  each encoding maps to a feasible solution
Example: binary vector Encoding

Given: graph
Goal: find minimum subset of nodes such that each edge is connected to at least one node of this subset
minimum vertex cover

<table>
<thead>
<tr>
<th>nodes selected?</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example: Integer Vector Encoding

**Given:**
- graph \( k \) colors

**Goal:**
- assign each node one of the \( k \) colors such that the number of connected nodes with the same color is minimized.

Graph Coloring Problem

<table>
<thead>
<tr>
<th>Nodes</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Colors</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

Diagram: A grid of nodes colored with different colors, illustrating the coloring problem.
Example: Real vector Encoding

\[ G^2(\vec{x}) = \left| \sum_{i=1}^{n} \cos^4(x_i) - 2 \prod_{i=1}^{n} \cos^2(x_i) \right| \]

parameters values
\[ x_1 \quad x \quad x \quad x \quad x_n \]

\[ \begin{array}{cccc}
. & . & 1. & . \\
\end{array} \]
Example: Parking a Truck

Objective: Find function $c$ with $u \quad c \quad x \quad y \quad d \quad t$
Search Space for the Truck Problem

Operators:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLUS(a,b)</td>
<td>returns a+b</td>
</tr>
<tr>
<td>MINUS(a,b)</td>
<td>returns a-b</td>
</tr>
<tr>
<td>MUL(a,b)</td>
<td>returns a*b</td>
</tr>
<tr>
<td>DIV(a,b)</td>
<td>return a/b, if b &lt;&gt; 0, else 1</td>
</tr>
<tr>
<td>ATG(a,b)</td>
<td>returns atan2(a,b), if a&lt;&gt;0, else 0</td>
</tr>
<tr>
<td>IFLTZ(a,b,c)</td>
<td>returns b, if a&lt;0, else returns c</td>
</tr>
</tbody>
</table>

Arguments:

- position x
- position y
- cab angle d
- trailer angle t

Search space: set of symbolic expression using the above operators and arguments
Encodes the function symbolic expression: $u \ x \ d \ y \ t$
A Solution Found by an EA

truck simulation

encoded tree
Design choices

- Representation
- Parameters
- Environmental selection
- Mating selection
- Variation operators
Fitness Assignment

Fitness \( F \) scalar value representing quality of an individual

The simple case:
- single objective optimization:

ore difficult cases:
- fitness not only takes into account the different objectives compliance to areto optimality but also properties of the whole population
- multiple optima need to be approximated diversity
- constraints are involved which have to be met
Simple example: areto Ranking

Fitness function:

\[ F(J) = \sum_{i=1, \ldots, N, J \neq J_i} \left\{ \begin{array}{ll} 1 & : J_i \prec J \\ 0 & : \text{else} \end{array} \right. \]

Execution time:

\[ F(1) = 3 \]
\[ F(2) = 1 \]
\[ F(3) = 1 \]
\[ F(4) = 2 \]
\[ F(5) = 1 \]
\[ F(6) = 0 \]
**Constraint Handling**

**Constraints:**

\[ x_1 \times x_n \geq \]

solution in solution space

**Approaches:**

- Construct initialization and variation such that infeasible solutions are not generated resp. not inserted.
- Representation is such that decoding always yields a feasible solution.
- Calculate constraint violation \[ x_1 \times x_n \] and incorporate it into fitness, e.g., penalty if \[ y \]
- Include the constraints as new objectives.
esign hoices

representation

fitness assignment

parameters

variation operators
Selection

Two types of selection:

- **mating selection** select for variation
- **environmental selection** select for survival
Tournament Selection

1. Uniformly choose individuals at random independently of fitness.
2. Compare fitness and copy best individual in mating pool.

Tournament selection binary tournament selection means
Design choices

representation - fitness assignment - mating selection

parameters

environmental selection
Vector mutation: Examples

- **it vectors:**
  - Each bit is flipped with probability 1

- **permutations:**
  - Swap
  - Rearrange
Mutation operators on Trees: rotation operators on Trees: growth
Mutation operators on Trees: Shrink

shrink

6 - 3
mutation operators on Trees: Sitch
mutation operators on Trees: Replace
ector Recombination: Examples

it vectors:

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
\end{array}
\]

permutations:

parents

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
1 & 1 & 1 & 1 \\
\end{array}
\]

child
Recombination of Trees
A generic ultimate effective EA

- Population
  - Sample
    - Vary
    - Select
  - New population
- Archive
  - Update
    - Truncate
  - New archive
STEP 1: Generate initial population and empty archive external set A t, et t.

STEP 2: Calculate fitness values of individuals in t and A t.

STEP 3: At 1 contains non-dominated individuals in t ∪ A t.
   If size of At 1 ≥ N then reduce At 1 else if size of At 1 < N then fill At 1 with dominated individuals in t and A t.

STEP 4: If t > to then output the non-dominated set of At 1.

STEP 5: Fill mating pool by binary tournament selection.

STEP 6: Apply recombination and mutation operators to the mating pool and set t 1 to the resulting population. et t t 1 and go to step 1.
dea Step: calculate dominance rank weighted by dominance count

Note: higher objective function better, smaller fitness better.
Course Synopsis

- Introduction
- Optimization
- Design
-
implementation: components

A framework that
▶ provides ready to use modules, algorithms, applications
▶ is simple to use
▶ is independent of programming language and O
▶ comes with minimum overhead

da: separate problem dependent from problem independent part
The concept of SA

Algorithms

Applications

A

N

A

A

knapsack

network processor design

text based platform and programming language independent interface for Search Algorithms [ler et al: ]
SA: mplementation

application independent:
- mating environmental selection
- individuals are described by state and objective vectors

handshake protocol:
- state action
- individual s
- objective vectors
- parameters

application dependent:
- variation operators
- stores and manages individuals
ard are Soft are odesign

mapping Applications To Architectures

doc dr regr apa
Synthesis

- Synthesis transforms behavior into structure.

  - : select components
  - : assign functions to components
  - *scheduling*: determine execution order

(mapping)

(allocation and) binding sometimes called *partitioning*
Application Specification

- Depends on the underlying model of computation.
- **Examples** (see also next slides):
  - Task graphs (data flow graph, control flow graph)
  - Process Networks (Kahn Process Network, Synchronous Dataflow)
  - State Machine Representations (SpecCharts, StateCharts, Polis) [not covered in this course].

- For the *mapping*, very often only the **network structure** and abstract properties of the processes are relevant (abstraction from detailed process function).
\[ x = 3a + b*b - c; \]
\[ y = a + b*x; \]
\[ z = b - c*(a + b); \]
what_is_this {
    read (a,b);
    done = FALSE;
    repeat {
        if (a>b)
            a = a-b;
        elseif (b>a)
            b = b-a;
        else done = TRUE;
    } until done;
    write (a);
}
an oce et o

- hierarchical network for M P application:
**Architecture Specification**

- Depends on the underlying model of the platform.
- Usually a graph notation is used to the elements, properties of the underlying platform are usually attached.
ample Architecture Specification

- <processor name="processor1" type="DSP">
  <port name="processor_port" type="duplex" />
  <configuration name="clock" value="100 MHz" />
</processor>

+ <processor name="processor2" type="RISC">
+ <memory name="sharedmemory" type="DXM">

- <hw_channel name="in_tile_link" type="bus">
  <port name="port1" type="duplex" />
  <port name="port2" type="duplex" />
  <port name="port3" type="duplex" />
  <configuration name="buswidth" value="32bit" />
</hw_channel>

- <connection name="processor1link">
  <origin name="processor1">
    <port name="processor_port" />
  </origin>
  <target name="in_tile_link">
    <port name="port1" />
  </target>
</connection>

+ <connection name="processor2link">
+ <connection name="memorylink">

7-
Mapping Specification

- Relates application and architecture specification:
  - maps processes to computing resources
  - maps communication between processes (in case of process networks) to communication paths of the architecture
  - specifies resource sharing disciplines and scheduling

Mapping = binding + scheduling
ample

asic model with a data flow graph and static scheduling

Data flow graph \( P(P, P) \)

Interpretation:
- \( V_P \) consists of functional nodes \( V_P^f \) (task, procedure) and communication nodes \( V_P^c \).
- \( E_P \) represents data dependencies.
Example (2)

Architecture graph $G_A(V_A, E_A)$:

- $V_A$ consists of functional resources $V_A^f$ (RISC, ASIC) and bus resources $V_A^c$. These components are potentially allocatable.
- $E_A$ models directed communication.
**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, $V_S=V_P \cup V_A$, $E_S=E_P \cup E_A \cup E_M$. 

---

**Example ( )**

Referring to the diagram:

1. Specification Graph $G_S$
2. Problem Graph $G_P$
3. Architecture Graph $G_A$
4. Edges $E_M$
5. RISC
6. SB
7. HWM1
8. PTP
9. HWM2
Example ( )

Three main tasks of synthesis:

• **Allocation** $\alpha$ is a subset of $V_A$.

• **Binding** $\beta$ is a subset of $E_M$, i.e., a mapping of functional nodes of $V_P$ onto resource nodes of $V_A$.

• **Schedule** $\tau$ is a function that assigns a number (start time) to each functional node.
**Definition**: Given a specification graph $G_S$ an implementation is a triple $(\alpha, \beta, \tau)$, where $\alpha$ is a feasible allocation, $\beta$ is a feasible binding, and $\tau$ is a schedule.
- Determine mapping
- Determine important parameters (end to end delay, throughput, buffer space output itter, )
- Give feedback to optimization

```
app  
pl a
  |
  |
  e
  |
  |
ema
```
- **low level**: at the register transfer (low) level, at the netlist level, split a digital circuit and map it to several devices (PGs, s)
  - stem parameters are relatively well known (area, delay)

- **high level**: at the system level, comparison of design alternatives is mandatory (design space exploration)
  - stem parameters are unknown
  - importance of estimation (analysis, simulation, rapid prototyping)
el

- (see previous lecture)
  
  model application
  define architectural template
  identify possible findings

- Often, parameters are attached to the architectural models that allow to simplify the partitioning (allocation and binding).
- Sometimes, (simulation, analysis) are applied to give more accurate predictions.

  Allocation gives cost as the sum of the allocated component costs.
  Scheduling gives latency.
  Constraints feasible schedule ≤ max feasible allocation ≤ max.
The partitioning problem is to assign \( n \) objects \( O = \{o_1, ..., o_n\} \) to \( m \) locations (also called partitions) \( P = \{p_1, ..., p_m\} \), such that:

- \( p_1 \cup p_2 \cup ... \cup p_m = O \)
- \( p_i \cap p_j = \forall i,j: i \neq j \) and
- cost \( c(P) \) are minimized

(simple model)

- objects data flow graph nodes
- locations architecture graph nodes
of a design point

- $C$ stem cost in
- $L$ latency in sec
- $P$ power consumption in

- requires to find $C$, $L$, $P$

linear cost function with penalty

$$f(C, L, P) = k_1 \cdot h_C(C, C_{max}) + k_2 \cdot h_L(L, L_{max}) + k_3 \cdot h_P(P, P_{max})$$

- $h_C$, $h_L$, $h_P$ denote how strong $C$, $L$, $P$ violate the design constraints $C_{max}$, $L_{max}$, $P_{max}$
- $k_1$, $k_2$, $k_3$ weighting and normalization
- enumeration
- Integer Linear Programs (ILPs)

- Constructive methods
  - random mapping
  - hierarchical clustering

- Iterative methods
  - Kernighan Lin Algorithm
  - simulated annealing

Evolutionary Algorithms (EAs) see next lecture
## Integer Programming o el

**Ingredients:**
- Cost function
- Constraints

In al in linear e pressions of inte er ariables from a set $X$

Cost function
\[ C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in \mathbb{R}, x_i \in \mathbb{N} \]  

Constraints:
\[ \forall j \in J : \sum_{x_i \in X} b_{i,j} x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R} \]

**Def.:** he problem of minimising (1) subject to the constraints (2) is called an integer programming (IP) problem.

If all $x_i$ are constrained to be either 0 or 1, the IP problem said to be a 0/1 integer programming problem.
ample

minimise \[ C = x + x + x \]

be t to \[ x_1 + x_2 + x \geq 2 \]

\[ x_1, x_2, x \in 0,1 \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( x_3 )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>15</td>
</tr>
</tbody>
</table>

optimal
on Integer Programming

- **aximizing** the cost function can be done by setting C C

- Integer programming is P-complete.

- In practice, running times can increase exponentially with the size of the problem, but problems of some thousands of variables can still be solved with commercial solvers, depending on the size and structure of the problem.

- IP models can be a good starting point for modeling, even if in the end heuristics have to be used to solve them.
Integer linear Program or Partitioning (1)

- Integer variables $x_i$
  - $x_i = 1$: object $i$ in bloc
  - $x_i = 0$: object $i$ not in bloc
- Cost $c_i$, if object $i$ is in bloc
- Integer linear program:
  
  \[
  x_{i,k} \in \{ \} \quad \leq i \leq n \quad \leq k \leq m
  \]

  \[
  \sum_{k=1}^{m} x_{i,k} = \quad \leq i \leq n
  \]

  \[
  \sum_{k=1}^{m} \sum_{i=1}^{n} x_{i,k} \cdot c_{i,k} \quad \leq k \leq m \quad \leq i \leq n
  \]
Integer Linear Program or Partitioning

- **Additional constraints**
  - Example: maximum number of objects in block
    \[ \sum_{i=1}^{n} x_{i,k} \leq h_k \quad \leq k \leq m \]

- The idea of mapping the synthesis problem to an IP is
  - **Er one ar**:
    - Scheduling can be integrated.
    - Various additional constraints can be added.
    - If not solved to optimality, run times are acceptable and a solution with a guaranteed quality can be determined.
    - Finding the right equations to model the constraints is an art.
random matching

- each object is assigned to a block randomly

Hierarchical clustering

- stepwise grouping of objects
- closeness function determines how desirable it is to group two objects

constructive methods

- are often used to generate a starting partition for iterative methods
- show the difficulty of finding proper closeness functions
Hierarchical Clustering - Example (1)

The diagram illustrates a hierarchical clustering example. The vertices $v_1$, $v_2$, $v_3$, $v_4$, and $v_5$ are connected with edges of different weights. The equation $v_5 = v_1 \cup v_3$ indicates that vertex $v_5$ is formed by the union of $v_1$ and $v_3$.

The closeness function used in this example is the arithmetic mean of weights.

- $v_1$ is connected to $v_2$ with weight 10, to $v_3$ with weight 20.
- $v_2$ is connected to $v_3$ with weight 10, to $v_4$ with weight 4.
- $v_3$ is connected to $v_4$ with weight 4.
- $v_4$ is connected to $v_5$ with weight 7.

The diagram visually represents the relationships and weights between the vertices.
Hierarchical Clustering - Example

\[ v = v_2 \cup v_5 \]
Hierarchical Clustering - Example ( )

\[ v_7 = v \cup v_4 \]
Hierarchical Clustering - Example ( )

ste :

ste :

ste :

V₁ V₂ V₃ V₄

8 - 1
Iterative Methods - ernighan- in (1)

- **simple greedy heuristic**:
  - until there is no improvement in cost: regroup a pair of objects which leads to the largest gain in cost

---

**Example:**
- before regroup: cost, number of edges crossing the partitions
- after regroup: gain
**Iterative Methods - ernighan- in ( )**

- **Problem**
  - Im le gree heuristic can get stuck in a local minimum

- **Improved algorithm** ernighan in:
  - As long as a better partition is found:
    - From all possible airs of objects virtually regroup the est lowest cost of the resulting partition then from the remaining not et touche eects virtually regroup the est air etc until all eects ha e een re grous
    - From these $n/2$ partitions take the one with smallest cost and actuall erform the corres on ing re grous o erations
iterative Methods - **simulated annealing**

- **from h sics**:
  - metal and gas take on a minimal energy state during cooling own under certain constraints:
    - at each temperature the system reaches a thermodynamic equilibrium
    - the temperature is increased sufficiently slow
  - a slight that a system can move to a higher energy state:
    \[
    P \left( e_i - e_{i+1} \right) \frac{1}{T_k} = e^{\frac{-e_i - e_{i+1}}{k_B T}}
    \]

- **application** to combinatorial optimization:
  - energy cost of a solution partition
  - cost decreases with temperature sometimes with a certain
    - a slight increases in cost are acceptable
Iterative Methods - Simulated Annealing

\[
\text{while} \quad \text{cost} \quad \text{c} \\
\text{while} \quad \text{random} \\
\{ \\
P' = \text{RandomMove}(P); \\
\text{cost}' = c(P'); \\
\text{deltacost} = \text{cost}' - \text{cost}; \\
\text{if} \ (\text{Accept}(\text{deltacost}, \text{temp}) > \text{random}[0,1]) \{ \\
\quad P = P'; \\
\quad \text{cost} = \text{cost}'; \\
\} \\
\} \\
\text{temp} = \text{DecreaseTemp}(\text{temp}); \\
\}
\]

\[
\text{Accept}(\text{deltacost}, \text{temp}) = e^{-\frac{\text{deltacost}}{k \cdot \text{temp}}}
\]
Iterative Methods - Simulated Annealing

- **Cooling Down**: DecreaseTemp(), Frozen()
  - temp_start = 1.0
  - temp = $\alpha \cdot$ temp  (typical: $0.8 \leq \alpha \leq 0.99$)
  - terminate when temp < temp_min or there is no more improvement

- **Equilibrium**: Equilibrium()
  - after defined number of iterations or when there is no more improvement

- **Complexity**
  - from exponential to constant, depending on the implementation of the functions Equilibrium(), DecreaseTemp(), and Frozen()
  - the longer the runtime, the better the quality of results
  - typical: construct functions to get polynomial runtimes
ard are Soft are odesign

Allo ation

do dr regor a a
Integer programming models

- ingredients:
  - cost function
  - constraints

Involving linear expressions of integer variables from a set

Cost function
\[ C = \sum_{x_i \in X} a_i x_i \text{ with } a_i \in \mathbb{R}, x_i \in \mathbb{N} \] (1)

Constraints:
\[ \forall j \in J : \sum_{x_i \in X} b_{i,j} x_i \geq c_j \text{ with } b_{i,j}, c_j \in \mathbb{R} \] (2)

Example: The problem of minimizing (1) subject to the constraints (2) is called an integer linear programming problem.

If all are constrained to be either 0 or 1, the P problem said to be an integer linear programming problem.
\[ = 1+ + \\
1+ + \geq \\
, , \in 0,1 \\
\]

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

optimal
Maximizing the cost function: use set

Integer programming is \text{P-complete}.

Running times depend exponentially on problem size, but problems of \( >1000 \) vars solvable with good solver (depending on the size and structure of the problem).

The case of \( \epsilon \in \mathbb{R} \) is called \( \text{P} \). \( \text{P} \) has polynomial complexity, but most algorithms are exponential, still in practice faster than for \( \text{P} \) problems.

The case of some \( \epsilon \in \mathbb{R} \) and some \( \epsilon \in \mathbb{N} \) is called

\( \text{P} \) \( \text{P} \) models can be a good starting point for modeling, even if in the end heuristics have to be used to solve them.
Simulated Annealing

- General method for solving combinatorial optimization problems.
- Based on the model of slowly cooling crystal liquids.
- Some configuration is subjected to changes.
- Special property of simulated annealing: changes leading to a poorer configuration (with respect to some cost function) are accepted with a certain probability.
- This probability is controlled by a temperature parameter: the probability is smaller for smaller temperatures.
**Initialization**

- Initially, some random initial configuration is created.
- Current temperature is set to a large value.

**Outer loop:**
- Temperature is reduced for each iteration
- Terminated if (temperature ≤ lower limit) or (number of iterations ≥ upper limit).

**Inner loop:** For each iteration:
- New configuration generated from current configuration
- Accepted if (new cost ≤ cost of current configuration)
- Accepted with temperature-dependent probability if (cost of new config. > cost of current configuration).
Multitjective timiation

- Maximize \((y_1, y_2, \ldots, y_k) = f(x_1, x_2, \ldots, x_n)\)

Pareto set = set of all Pareto-optimal solutions
Summary

- Single objective optimization methods
  - decision is performed during optimization
  - Examples: integer programming, simulated annealing

- Multiple objective optimization methods
  - decision is done after optimization
  - Example: Evolutionary algorithms
  - Refer to publications of Thiele or Schwefel et al. for more information

- Concept of Pareto points
  - eliminates large set of non-relevant design points
  - allows separating optimization and decision
- oop caches
- Mapping code to less used part(s) of the index space
- Cache locking freezing
- Changing the memory allocation for code or data
- Mapping pieces of software to specific ways

Methods:
- generating appropriate way in software
- allocation of certain parts of the address space to a specific way
- including way-identifiers in virtual to real-address translation

Caches behave almost like a scratch pad
Summary

- Allocation strategies for SPM
  - Dynamic sets of processes
  - Multiprocessors
  - MMs
  - Sharing between SPMs in a multi-processor
- Optimizations for Caches
  - Code layout transformations
  - Ray prediction
Granularity: size of tasks (e.g. in instructions)
Readable specifications and efficient implementations can possibly require different task structures.
Granularity changes
- Reduced overhead of context switches,
- More global optimization of machine code,
- Reduced overhead for inter-process task communication.
- less blocking of resources while waiting for input,
- more flexibility for scheduling, possibly improved result.
The most appropriate task graph granularity depends upon the context, and merging and splitting may be required.

Merging and splitting of tasks should be done automatically, depending upon the context.
PROCESS GetData
    (InPort IN, OutPort DATA){
        float sample, sum; int i;
        while (1) {
            sum=0;
            for (i=0; i<N; i++){
                READ(IN, sample, 1)
                sum+=sample;
                WRITE(DATA, sample, 1)
            }
            WRITE(DATA, sum/N, 1);
        }
    }

PROCESS Filter(InPort DATA,
    InPort COEF, OutPort OUT){
    float c,d; int j;
    c=1; j=0;
    while(1) {
        SELECT(DATA, COEF){
            case DATA: READ (DATA, d, 1);
                if (j==N){j=0; d=d*c; WRITE(OUT, d, 1);
                    } else j++;
                    break;
            case COEF: READ(COEF, c, 1); break;
        }
    }}}}
Attributes of a system that sees

Tasks blocking after they have already started running

```c
PROCESS GetData
  (InPort IN, OutPort DATA)
  float sample, sum; int i;
  while (1) {
    sum = 0;
    for (i = 0; i < N; i++){
      READ(IN, sample, 1)
      sum += sample;
      WRITE(DATA, sample, 1)
    }
    WRITE(DATA, sum/N, 1);
  }

PROCESS Filter(InPort DATA,
              InPort COEF, OutPort OUT)
  float a, c, d; int j;
  c = 1; j = 0;
  while (1), {
    SELECT(DATA, COEF)
    case DATA: READ (DATA, d, 1);
    if (j == N) j = 0; d = d*c; WRITE(OUT, d, 1);
    } else j++;
    break;
    case COEF: READ(COEFF, c, 1); break;
  }
```
or y orta e a et a

1. Transform each of the tasks into a Petri net,
2. Generate one global Petri net from the nets of the tasks,
   - Partition global net into sequences of transition
   - Generate one task from each such sequence

Mature, commercial approach not yet available
es
tasu s
ey ortha e a

Rads only at the beginning

initialization task

Init()
sum=0;i=0;c=1;j=0;
}

Tin()
READ(IN,sample,1);
sum+=sample; i++;
DATA=sample; d=DATA;
if (j==N) {j=0; d=d*c; WRITE(OUT,d,1);
} else j++;
L0: if (i<N) return;
DATA=sum/N; d=DATA;
if (j==N) {j=0; d=d*c; WRITE(OUT,d,1);
} else j++;
sum=0; i=0; goto L0

always true

COEF

always true
ever true

Tin()
READ(IN, sample, 1);
sum+=sample; i++;
DATA=sample; d=DATA;
if (j==N) {j=0; d=d*c; WRITE(OUT,d,1);
} else j++;
L0: if (i<N) return;
DATA=sum/N; d=DATA;
if (j==N) {j=0; d=d*c; WRITE(OUT,d,1);
} else j++;
sum=0; i=0; goto L0

always true

T = sample  d =

T = sample  d =

(i < ) return

T = sum  d =

d = d c  R TE(  T,d,1)

sum =  i =

return
The dynamic behavior of applications getting more attention.
Energy consumption reduction is the main target.
Some classes of applications (i.e. video processing) have a considerable variation in processing power requirements depending on input data.
Static design-time methods becoming insufficient.
Runtime-only methods not feasible for embedded systems.

- ow about mixed approaches
Static (compile-time) methods can ensure CET feasible schedules, but waste energy in the average case. …or they can define a probability for violating the deadline.

Mixed methods use compile-time analysis to define a set of possible execution parameters for each task. Runtime scheduler selects the most energy saving, deadline preserving combination.
Pros:
- lower cost
- aster
- lower power consumption
- Sufficient S R, *if properly scaled*
- Suitable for portable applications

Cons:
- increased dynamic range
- finite word-length effect, *unless properly scaled*
- Overflow and excessive quantization noise
- Extra programming effort
floating-Point vs. fixed-Point

- *exponent*, mantissa
- floating-Point: automatic computation and update of each exponent at run-time
- fixed-Point: implicit exponent determined off-line

© Ki-Il Kum, et al
ss me t a
to Su tra to

- Assume $y = x$, with
  - $x (\ = 2)$ and
  - $y (\ = )$:

$$\text{result} = x \ y$$

eualizing each

© Ki-II Kum, et al
>>> assume result = x \ y, with
    \ - x ( =2) and
    \ - y ( = )
    \ - result ( =2 )

© Ki-Il Kum, et al
Welcome to Pot Program

- oat Pot Program
- ee Pot Program
- e est mat o Program
- aaua Pot Program

© Ki-Il Kum, et al
float iir1(float x)
  static float s =
  float y
  y = . s x
  range(y, 0);
  s = y
  range(s, 1);
  return y
erat o s  e o t ro ram

- x 2^1
  - s
    - iwl= .xxxxxxxxxxxxx

- overflow if ≠

- result

- x
  - iwl= .xxxxxxxxxxxxx
xed-Point C Program

```c
int iir1(int x)
{
    static int s =
    int y
    y = sll(mulh(29491, s) + (x >> 5), 1);
    s = y
    return y
}
```

- **mulh**
  - to access the upper half of the multiplied result
  - target dependent implementation
- **sll**
  - to remove 2\textsuperscript{nd} sign bit
  - opt. overflow check

© Ki-Il Kum, et al
Performance of arithmetic operations

Cycles

P

1 1

12 2

2 1

2 1

ixed-Point (1b) ixed-Point (2b) loating-Point

© Ki-II Kum, et al
Array

Row major order (C)

Column major order (RTR)
est er orma e ermost oo orres o s to r tmost array e

- o oo s assum ro ma or or er
- or (k= k<=m k ) or (j= j<=n j )
- or (j= j<=n j ) or (k= k<=m k )
- p j k = ... p j k = ...

Same behavior for homogenous memory access, but:

- or row major order

↑ Poor cache behavior ood cache behavior ↑

☞ memory architecture dependent optimization
Example:
...#define iter 400000
int a[20][20][20];
void computeijk() {int i,j,k;
    for (i = 0; i < 20; i++) {
        for (j = 0; j < 20; j++) {
            for (k = 0; k < 20; k++) {
                a[i][j][k] += a[i][j][k];}
        }
    }
}
void computeikj() {int i,j,k;
    for (i = 0; i < 20; i++) {
        for (j = 0; j < 20; j++) {
            for (k = 0; k < 20; k++) {
                a[i][k][j] += a[i][k][j];}
        }
    }
}
start=time(&start);for(z=0;z<iter;z++)computeijk();
end=time(&end);
printf("ijk=%16.9f\n",1.0*difftime(end,start));
(S interchanges array indexes instead of loops)
Root structure: i j

ramatic impact of locality

<table>
<thead>
<tr>
<th>Professor</th>
<th>Su</th>
<th>SP</th>
<th>Pet um</th>
</tr>
</thead>
</table>

not always the same impact..

Tilluchwald, diploma thesis, niv.ortmund, informatik 12, 122
or (j = j <= n j ) or (j = j <= n j )

pj = ... pj = ...

or (j = j <= n j )
pj = pj ...

oops small enough to allow zero overhead access to p.

eter locality for etter chances for parallel execution.

hich of the two versions is best archiecture-aware compiler should select best version.
```c
#define size 30
#define iter 40000
int a[size][size];
float b[size][size];

void ss1() {int i, j;
    for (i=0; i<size; i++) {
        for (j=0; j<size; j++) {
            a[i][j] += 17;
        }
    }
    for (i=0; i<size; i++) {
        for (j=0; j<size; j++) {
            b[i][j] -= 13;
        }
    }
}

void ms1() {int i, j;
    for (i=0; i<size; i++) {
        for (j=0; j<size; j++) {
            a[i][j] += 17;
        }
    }
    for (j=0; j<size; j++) {
        for (i=0; i<size; i++) {
            b[i][j] -= 13;
        }
    }
}

void mm1() {int i, j;
    for (i=0; i<size; i++) {
        for (j=0; j<size; j++) {
            a[i][j] += 17;
            b[i][j] -= 13;
        }
    }
}
```
with Sparc except Sparc loops.

**Merge**

- gc2 - x
gcc 2.0 Sparc gcc x

P amet orm

Merge d loops

superior except Sparc with o
- or \( j = j \leq n \)  
  - \( p_j = ... \)

- or \( j = j \leq n \) = 2  
  - \( p_j = ... \)  p_{j + 1} = ...

factor = 2  
- better locality for access to \( p \).
- fewer branches per execution of the loop. More opportunities for optimizations.
- Tradeoff between code size and improvement.
- Extreme case: completely unrolled loop (no branch)
```c
#define s 30
#define iter 4000
int a[s][s], b[s][s], c[s][s];
void compute(){
    for(i=0;i<s;i++){
        for(j=0;j<s;j++){
            for(k=0;k<s;k++){
                c[i][k] += a[i][j]*b[j][k];
            }
        }
    }
}

extern void compute2()
    {
    int i, j, k;
    for (i = 0; i < 30; i++) {
        for (j = 0; j < 30; j++) {
            for (k = 0; k <= 28; k += 2) {
                int *suif_tmp; 
                suif_tmp = &c[i][k];
                *suif_tmp = *suif_tmp + a[i][j]*b[j][k];
            }
        }
    }
    return;
    }
```
- The benefits of using a processor with suitable small penalties may be large.

Till uchwald, diploma thesis, niv. ortmund, nformatik 12, 12 2
```c
#define s 50
#define iter 150000
int a[s][s], b[s][s];
void compute() {
    int i,k;
    for (i = 0; i < s; i++) {
        for (k = 1; k < s; k++) {
            a[i][k] = b[i][k];
            b[i][k] = a[i][k-1];
        }
    }
}
```

- Small benefits
or \( i = 1 \leq i \leq i \)

or \( k = 1 \leq k \leq k \)

\( r = i,k \) to be allocated to a register

or \( j = 1 \leq j \leq j \)

\( i,j = r \leq k,j \)

ever reusing information in the cache for and if is large or cache is small (2 references for ).
or (kk=1  kk<=  kk = )
or (jj=1  jj<=  jj = )
or (i=1  i<=  i )
or (k=kk  k<=  min(kk -1,  )  k )
or (j=jj  j<=  min(jj -1,  )  j )

Compiler should select best option

Same elements for next iteration of i
The extracted text is not fully legible, but it appears to be discussing results and may involve rates or similar metrics. The diagram shows a bar graph with two labels: "SP" and "PETUM." The x-axis seems to represent different values, and the y-axis shows a logarithmic scale from 0.00 to 1000.00. The legend indicates that this graph is from Tilluchwald's diploma thesis at Dortmund University, Department of Informatics.
Summary

- Task concurrency management
  - Re-partitioning of computations into tasks
  - Dynamic exploitation of slack
- Floating-point to fixed point conversion
  - Range estimation
  - Conversion
  - Analysis of the results
- High-level loop transformations
  - Unrolling
  - Tiling
many if-statements for margin-checking \rightarrow \text{no checking, efficient} \rightarrow \text{only few margin elements to be processed}
oop nest from MPE - full search
motion estimation

for (z= z 2 z )
for (x= x x ) x1= x
for (y= y y ) y1= y
for (k= k k ) x2=x1 k-
for (l= l ) y2=y1 l-
for (i= i i ) x =x1 i x =x2 i
for (j= j j ) y =y1 j y =y2 j
if (x x y y )
then block 1 else else block 1
if (x x y y )
then block 2 else else block 2

analysis of polyhedral domains,
selection with genetic algorithm

for (z= z 2 z )
for (x= x x ) x1= x
for (y= y y )
if (x =1 y =1 )
for ( y y )
for (k= k k )
for (l= l l )
for (i= i i )
for (j= j j )
then block 1 then block 2
else y1= y
for (k= k k ) x2=x1 k-
for (l= l ) y2=y1 l-
for (i= i i ) x =x1 i x =x2 i
for (j= j j ) y =y1 j y =y2 j
if (x x y y )
then block-1 else else block-1
if (x x y y )
then block 2 else else block 2

alk et al., nf 12, nio, 2 2
Motion Estimation

alk et al., nf 12, nio, 2 2
rray o

- initial arrays
rray o

- nfolded arrays
Array folding is implemented in the TSE optimization proposed by MEC. Array folding adds div and mod ops. Optimizations required to remove these costly operations.

In MEC, PT address optimizations perform this task. For example, modulo operations are replaced by pointers (indexes) which are incremented and reset.

```c
for(i=0; i<20; i++)
    B[i % 4];

tmp=0;
for(i=0; i<20; i++)
    if(tmp >= 4)
        tmp -= 4;
    B[tmp];
    tmp ++;
```

```plaintext
\[ i \mod 4 \quad \text{if} (\text{tmp} \geq 4) \quad \text{tmp} = \text{tmp} - 4 \quad \text{tmp} = \text{tmp} + 1 \]
```
required to achieve real benefit

Prilagoditev kode

- prenos zapisa iz ANSI-C v Handel-C
  - VHDL zahteva bistveno več sprememb

- opis algoritma v C kodni je treba pred strojno izvedbo ustrezno prilagoditi
  - SystemC oz. Handel-C vsebuje samo podmnožico ukazov običajnega C
  - drugače je treba realizirati aritmetiko plavajoče vejice, ki je strojne izvedbe načeloma ne podpirajo
    - zavzame preveč razpoložljivih virov
    - zmanjšuje frekvenco delovanja
  - vnos ukazov za vzporedno izvajanje delov kode
  - prilagoditev velikosti vseh spremenljivk
Prilagoditev rogra ke kode

- nadomestek aritmetike plavajoče vejice
  - uporaba fiksne vejice
  - uporaba celoštevilčnih vrednosti manjša enota mere

- vrednosti s fiksno vejico so pomnožene in predstavljene kot celoštevilke vrednosti
  
<table>
<thead>
<tr>
<th>var</th>
<th>var2</th>
</tr>
</thead>
<tbody>
<tr>
<td>signed int 6 si</td>
<td>signed int 6 si</td>
</tr>
<tr>
<td>si 0x0 a0</td>
<td>si .62</td>
</tr>
<tr>
<td>var si [ : ]</td>
<td>var 0x0</td>
</tr>
</tbody>
</table>
  | var2 si [ :0]  | var2 0x0a0    | 60
Prilagoditev rogra ke kode

- ukazi za vzporedno izvajanje delov kode
  - ukaz namesto
    - kjer je mogoče, glede na vsebino zanke

for i 0 i 3 i
  a[i] b[2 i]
se
  a[i] b[2 i]
a[i] a[i] c[i]
b[2 i] a[i]

par i 0 i 3 i
  a[i] b[2 i]
se
  par
  a[i] b[2 i]
a[i] a[i] c[i]
b[2 i] a[i]
**Prilagoditev rogra ke kode**

- prilagoditev velikosti vseh spremenljivk
  - vse velikosti morajo biti vnaprej definirane
    - za manjšo porabo virov naj bodo minimizirane
  - vnaprej je treba določiti predznačene nepredznačene
  - pri računanju s spremenljivkami različnih velikosti
    - uporaba operatorja spajanja: manjši spremenljivki dodamo manjkajoča mesta
    - uporaba spodnjih mest pri večji spremenljivki

```
[signed  unsigned] int n   n-bit
unsigned int  6 var , var3
unsigned int   var2, var
var3   var       var2
var   var        var2
```

10 - 1
ard are o t are ode ig

o ilatio

do dr regor Pa a
Many reports about low efficiency of standard compilers

- Special features of embedded processors have to be exploited.
- High levels of optimization more important than compilation speed.
- Compilers can help to reduce the energy consumption.
- Compilers could help to meet real-time constraints.

Less legacy problems than for PCs.

- There is a large variety of instruction sets.
- Design space exploration for optimized processors makes sense
Energy Access times

Access times

Energy

Memory Size (Bytes)

64 128 256 512 1k 2k 4k 8k 16k 32k 64k

Energy [nJ]
Access Time [ns]
• High-performance if available memory bandwidth fully used
• low-energy consumption if memories are at stand-by mode
• reduced energy if more values are kept in registers

LD r3, [r2, 0]
ADD r3, r0, r3
M V r0, 2
LD r0, [r2, r0]
ADD r0, r3, r0
ADD r2, r2,
ADD r, r,
CMP r, 00
LT LL3

int a[000]
c a
for i i 00 i
b c
b c
c
ADD r3, r0, r2
M V r0, 2
M V r2, r 2
M V r 2, r
M V r , r r 0
M V r0, r
M V r, r
M V r , r
LD r , [r , r0]
ADD r0, r3, r
ADD r , r ,
ADD r, r ,
ADD r, r ,
CMP r , 00
LT LL3
- Energy-aware scheduling
- Energy-aware instruction selection
- Operator strength reduction: e.g. replace $a[i]$ by $a[i-1]$
- Minimize the bitwidth of loads and stores
- Standard compiler optimizations with energy as a cost function

E.g.: register pipelining:

```plaintext
for i: 0 to 0 do
begin
  C: 2  a[i]  a[i-]
end
```

- Exploitation of the memory hierarchy
AM TDMI cores, well-known for low power consumption

Hierarchy

- main
- SPM
- processor

no tag memory

Example

Address space

- scratch pad memory
er limited ort i
a ed tool lo

e rag a i o r e to allo ate to e i i e tio

or example:
#pragma arm section rwd = "foo", rodata = "bar"
int x2 = 5; // in foo (data part of region)
int const z2[3] = {1,2,3}; // in bar


t a t t e l oad i g ile to li ker or allo ati g e tio to

e i i addre ra ge

LOAD_ROM_1 0x0000
{
  EXEC_ROM_1 0x0000
  {
    program1.o (+RO)
  }
  DRAM 0x10000 0x8000
  {
    program1.o (+RW,+ZI)
  }
}

http: www.arm.com documentation
Software Development Tools index.html
Example:

Which memory object array, loop, etc. to be stored in SPM

\[ G = \sum g_k \]

for each segment \( k \). Maximise gain \( G = \sum g_k \), respecting size of SPM SSP \( \geq \sum s_k \).

Solution: knapsack algorithm.

Moving objects back and forth
Preemption 
Migration and Variable Size

- \( S var_k \): size of variable \( k \)
- \( n_k \): number of accesses to variable \( k \)
- \( e var_k \): energy _averaged_ per variable access, if \( var_k \) is migrated
- \( E var_k \): energy _averaged_ if variable \( var_k \) is migrated \( e var_k n \ var_k \)
- \( x var_k \): decision variable, if variable \( k \) is migrated to SPM, 0 otherwise

\( K \): set of variables

Similar for functions \( I \)

**Integer Programming**

Maximize \( \sum_{k \in K} x var_k E var_k \sum_{i \in I} x F_i E F_i \)

Subject to the constraint
\( \sum_{k \in K} S var_k x var_k \sum_{i \in I} S F_i x F_i \leq SSP \)
ed time energy and average runtime

Multi sort mix of sort algorithms

Numbers will change with technology, algorithms remain unchanged.

Measured processor external memory energy
CACTI values for SPM combined model
Fine-grained granularity smoothens dependency on the size of the scratch pad.

This requires additional jump instructions to return to main memory.

Statically 2 jumps, but only one is taken or consecutive basic blocks.
Estimated generation of additional jumps with special compiler.
A slide showing a graph with various data points indicating energy savings in percentages plotted against total scratchpad capacity in bytes. The graph has multiple lines, each representing different models or methods, such as Multi_Sort, Encode_combined, Fast_idct, FFT_Viva, and ref_idct. The graph is labeled 'Combined model for memories.'
aiT:
- WCET analysis tool
- support for scratchpad memories by specifying different memory access times
- also features experimental cache analysis for AM
A M TDMI with 3 different memory architectures:

ai e or
LD -cycles: CP ,l ,D 3,2,2
ST -cycles: 2,2,2
  ,2,0

ai e or i ied a e
LD -cycles: CP ,l ,D 3, 2,6
ST -cycles: 2, 2,3
  ,2,0

ai e or rat ad
LD -cycles: CP ,l ,D 3,0,2
ST -cycles: 2,0,0
  ,0,0
Influence of nchip Scratchpad Memories on WCET: th Intl Workshop on worst-case execution time WCET analysis, Catania, Sicily, Italy, une 2 , 200

Second paper on SP Cache and WCET at DATE, March 200
scratch pad 0, 256 entries
scratch pad 1, 2 k entries
scratch pad 2, 16 k entries
"main" memory
Minimize $C = \sum_{j} e_{j} \cdot \sum_{i} x_{j,i} \cdot n_{i}$

With $e_{j}$: energy per access to memory $j$,
and $x_{j,i}$: if object $i$ is mapped to memory $j$, 0 otherwise,
and $n_{i}$: number of accesses to memory object $i$,
subject to the constraints:

\[ \forall j : \sum_{i} x_{j,i} \cdot S_{i} \leq SSP_{j} \]

\[ \forall i : \sum_{j} x_{j,i} = 1 \]

With $S_{i}$: size of memory object $i$,
$SSP_{j}$: size of memory $j$. 
Table 1: Example of all considered memory partitions for a total capacity of 4096 bytes
A key advantage of partitioned scratchpads for multiple applications is their ability to adapt to the size of the current working set.
Effectively results in a kind of "troller" for SPM.

Address assignment within SPM required paging or segmentation-like.

Reference: Verma, Marwedel: Dynamic overlay of Scratchpad Memory for Energy Minimization, ISSS 200
rat  ad  a ed o  live e
a al  i

- M  A, T , T2, T3, T
- SP Size  A  T
  T

P

P

P

A  SP & T3  SP

P
High-level transformations
- Loop nest splitting
- Array folding

Impact of memory architecture on execution times & energy.

The SPM provides
- Uptime efficiency
- Energy efficiency
- Timing predictability

Achieved savings are sometimes dramatic, for example:
- Savings of \(\frac{1}{3}\) of the memory system energy
ard are o t are ode ig

Per or a e ti atio

do dr regor Pa a
The values of the *objective functions* that should guide the design space exploration are obtained through

Design space exploration intends to change
- mapping, binding, and resource sharing
- architecture, hardware platform
- application choice between different algorithms and/or partitioning into concurrent components
Performance Metrics

Subsystems

Abstraction Levels

Performance Estimation Methods
PERFORMANCE ESTIMATION METHOD:

METRIC
Other: Quality, SNR,...
Cost
Area
Power
Time

ABSTRACTION LEVEL
High-level e.g. functional, HLL
Intermediary level e.g. TLM, OS
Low-level e.g. RTL, ISA

Note:
RTL – Register Transfer Level
ISA – Instruction Set Architecture
TLM – Transaction-Level Model
OS – Operating System
HLL – High-Level Language

x(y) = x0 * exp(-k0*y)
x0 = 105
k0 = 1.2593
Advantages: short simulation time, no details of implementation necessary

Drawbacks: limited accuracy, e.g. no information about timing

Advantages: higher accuracy

Drawbacks: long simulation time, many implementation details need to be known
Prerequisite for

- part of the feedback cycle see global flow
- functional and non-functional validation e.g. power, energy, timing, memory consumption

- show equivalence of specification and implementation
- functional and non-functional aspects
Overview

Performance Estimation Methods

Subsystems

Abstraction Levels
Per or a e ti atio lo al Pi t re

PERFORMANCE ESTIMATION METHOD

ABSTRACTION LEVEL

METRIC
Other, Quality, SNR, ... Cost Area Power Time

simulation statistic analytic

x(y) = x0 * exp (-k0* y)
x0 = 105
k0 = 1.2593

HW subsystem

CPU subsystem

SW subsystem

interconnect subsystem

HW IP

SW ss.

MPSoC

communication

Low-level e.g. RTL, ISA

Intermediate level e.g. TLM, OS

High-level e.g. functional, HLL

SUBSYSTEM TO ANALYZE

Note:
RTL – Register Transfer Level
ISA – Instruction Set Architecture
TLM – Transaction-Level Model
OS – Operating System
HLL – High-Level Language
Performance Metrics

- **Performance metric** = function defined on relevant non-functional properties of a system which indicates a quantitative performance of the system.

- **Time** [second]
  - for example end-to-end delay, throughput, latency

- **Power, Energy, Temperature** [mW, mJ, °C]
  - for example power consumed by the network, energy execute a task, maximal temperature

- **Area** [mm²]
  - for example area of an integrated circuit

- **Cost** [$]
  - for example cost of parts, labor, development cost

- **Other metrics:**
  - SNR (signal to noise ratio), quality of the video image/sound, size of the hardware platform

usually, performance metrics are conflicting
<table>
<thead>
<tr>
<th>Example of Performance</th>
<th>ra</th>
<th>effs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main Domain</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ change the mapping of the application to the architecture</td>
<td></td>
<td>see example 1</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture Domain</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ change the hardware platform</td>
<td></td>
<td>see example 2</td>
</tr>
</tbody>
</table>

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Application Domain</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>▪ change the application implementation (e.g. degree of parallelization, partitioning into concurrent processes, use of different algorithms with a similar functional behavior)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Era e ffs in t e Ma in omain

- **PE apping Optimization**
  - 2 mapping optimization space
    - ob Worst load of computation node
    - ob 2 Worst load of communication node

![Diagram](image)

worst bus load
Effective performance in the era are Platform

- General purpose processors
- Instruction set specific processors (Ps)
- Microcontroller
- Programmable arrays
- Specific circuit implementations

Energy efficiency

Microcontroller

Programmable arrays

Circuit implementations
Outline

- Overview
- Performance Metrics
- Ubs stems
- Abstraction Levels
- Performance Estimation Methods
Performance Estimation – Global Picture

PERFORMANCE ESTIMATION METHODS

METRIC
Other: Quality, SNR, ... Cost
Area
Power
Time
Simulation
Statistic
Analytic

ABSTRACTION LEVEL

M1 M2 ... interface HW subsystem
CPU subsystem
SW subsystem
Co-interconnect subsystem
MPSOC

SUBSYSTEM TO ANALYZE

METRIC
Other: Quality, SNR, ... Cost
Area
Power
Time
Simulation
Statistic
Analytic

x(y) = x0 * exp(-k0*y)

x0 = 105
k0 = 1.2593

Note:
RTL – Register Transfer Level
ISA – Instruction Set Architecture
TLM – Transaction-Level Model
OS – Operating System
HLL – High-Level Language
Communication emulates

omutation emulates

rc architecture

c e ulin an rbitation

emulates

E E

tional

M

are

static

namic
i e

riorit

riorit
Estimation difficult

- **Computation and communication**
  - (Non-deterministic) computations in processing nodes
  - (Non-deterministic) communication delays
  - Complex resource interaction via scheduling and arbitration policies

- **Cyclic timing encodings**
  - Internal data streams interact on computing and communication resources
  - Interaction determines stream characteristics

- **Uncertain environment**
  - Different load scenarios
  - Unknown (worst case) inputs
Illustration of Evaluation Difficulties

as communication
as c e ulin
om le n ut
imin itter bursts
ifferent E ent es
Illustration of Evaluation Difficulties

Processor as a sufferer in the communication
of completions as communication

variable as resource availability
variable execution emanating

as communication
as current line
omlementary
internal state program access

variable stream
variable different events

stream
stream different events
stream

ab acc b
Requirements for Performance Estimation

- Estimation should be **composable** in terms of:
  - *software* systems and their *interactions*, i.e., W, SW, interconnect
  - *computation, communication*, and *scheduling iteration*

- Estimation should cover different *metrics*, for example power, energy, delay, memory, throughput

- Estimation method should represent a *reasonable trade-off* between (a) estimation effort in terms of computation/simulation time and set-up time and (b) accuracy
Outline

- Overview
- Performance metrics
- Subsystems
  - Abstraction elements
  - Performance estimation methods
Performance Estimation – Global Picture

PERFORMANCE ESTIMATION METHODS

METRIC
Other: Quality, SNR …
Cost
Area
Power
Time

SW subsystem
communication
CPU subsystem
HW subsystem

Interface HW subsyste...

SUBSYSTEM TO ANALYZE

HW IP
SW ss.
SW ss.
SW ss.

x(y) = x0 \times \exp(-k0y)
\begin{align*}
x0 &= 105 \\
k0 &= 1.2593
\end{align*}

Note:
RTL – Register Transfer Level
ISA – Instruction Set Architecture
TLM – Transaction-Level Model
OS – Operating System
HLL – High-Level Language

Function:
\begin{align*}
x(y) &= x0 \times \exp(-k0y) \\
x0 &= 105 \\
k0 &= 1.2593
\end{align*}
Register-transfer level model

(data[ ] (critical path latency)

Transaction

SW, W systems

to ens. comm. backbones,

SW tasks, W/SW codes/cosim. tools

formal methods

s

R

s

s

s

2

s

2

transistor model

(t=RC)

technology

transistors, layouts

signal
gate, schematic, R

simulator S C

simulator SystemC

simulator

SystemC

simulator

SystemC

simulator
Outline

- Overview
- Performance Metrics
- Subsystems
- Abstraction Levels
- Performance Estimation Methods
Performance Estimation – Global Picture

PERFORMANCE ESTIMATION METHOD

METRIC
Other, Quality, SNR, Cost, Area, Power, Time

ABSTRACTION LEVEL

Performance estimation methods can be categorized based on the level of abstraction, METRIC, and SUBSYSTEM TO ANALYZE. The graph illustrates the relationship between these factors.

- **METRIC**: Time, Cost, Area, Power, Other (Quality, SNR, …)
- **ABSTRACTION LEVEL**: Subsystem level, HW IP, communication, SESE (CPU), HW itf., SW ss., SW ss., SW ss., MPSoC

The performance estimation methods include:
- **Analytic**: x(y) = x0 * exp(-k0*y), with x0 = 105 and k0 = 1.2593
- **Simulation**: Simulation approach
- **Statistic**: Statistical analysis

**SUBSYSTEM TO ANALYZE**

- HW subsystem
- SW subsystem
- CPU subsystem
- Interconnect subsystem

**PERFORMANCE ESTIMATION METHODS**

- **CPU subsystem**: Analytic, Simulation, Statistic
- **Interconnect subsystem**: Analytic, Simulation, Statistic
- **SW subsystem**: Analytic, Simulation, Statistic
- **CPU**: Analytic, Simulation, Statistic

Note:
- **HW IP**: Register Transfer Level (RTL), Instruction Set Architecture (ISA)
- **TLM**: Transaction-Level Model
- **OS**: Operating System
- **HLL**: High-Level Language

**SUBSYSTEM TO ANALYZE**

- HW IP
- SW ss.
- SW ss.
- MPSoC

**Communication**

- Low-level: e.g., RTL, ISA
- Intermediary level: e.g., TLM, OS
- High-level: e.g., functional, HLL

**Communication**

- Task 1
- Task 2
- Task 3

**Note:**
- RTL – Register Transfer Level
- ISA – Instruction Set Architecture
- TLM – Transaction-Level Model
- OS – Operating System
- HLL – High-Level Language
e.g. delay

Real System

Worst-Case

Best-Case

Measurement

Simulation

Probabilistic Estimation

Worst Case (Formal) Analysis

→ presented in Lecture 6 (next lecture)

→ presented later
To evaluate the system, one can use measurements, formal analysis, simulation, or statistics. Each method involves different steps:

- **Measurements**: Use an existing instance of the system to perform performance measurements.

- **Formal Analysis**: Develop a mathematical abstraction of the system and derive formulas to describe the system's performance.

- **Simulation**: Develop a program that implements a model of the system. Perform experiments by running the program.

- **Statistics**: Develop a statistical abstraction of the system and derive statistical performance analysis or simulation.
Performance Estimation Methods

- 
- designers experience
- component simulation
- model o application
- system model
- model o architecture
- estimation tool (method)
- estimation results
- input traces
- spec. o inputs
- data sheets
- platform benchmarks
nalytic Models

- **Static analytic symbolic models**
  - Describe computing communication and memory resources by algebraic equations e.g.
    
    \[
    \text{delay} = \left\lfloor \frac{\# \text{words}}{\text{burst size}} \right\rfloor \times \text{comm time}
    \]

  - Describe system properties by parameters e.g. data rate
  - Combine relations

- Fast and simple estimation
- Generally inaccurate modeling e.g. resource sharing not modeled
Dynamic Analytic Models

- Combination between
  - **Static models** possibly extended by non-determinism in run-time and event processing
  - **Dynamic models** or describing e.g. resource sharing mechanisms (scheduling and arbitration).

- Existing approaches
  - Theory
  - (statistical bounds)
  - Worst case best case
Exam le - e in Systems

- clients request some service from a server over a network.
- Performance of the server
- Performance of the network

![Diagram of a client-server network with a server and network (Internet)]
Stochastic Models - Queuing Systems

- **queuing system** is described by:
  - arrival rate
  - Service mechanism
  - queuing discipline

- **Performance measures**:
  - average delay in queue
    - Customer point of view
  - time-average number of customers in queue.
    - System point of view
  - proportion of time server is busy

The classical M/M/1 queuing system:
(M = Markovian (exp.) distribution)

![Diagram of M/M/1 queuing system](image)
**ondete mimistic Models - Queuing Systems**

- A *queuing system* is described by:
  - Arrival function (bounds on arrival times)
  - Service functions (bounds on server behavior)
  - Resource interaction

- **Performance measures**:
  - Worst case delay in queue
  - Worst-case number of customers in queue
  - Worst-case and best-case end-to-end delay in the system

![Diagram of queuing system]
Simulation

- Consider the underlying hardware platform and the mapping of the application onto that architecture
- Combine functional simulation and performance data
- Evaluate average-case behavior for one simulation scenario
- Complex set-up and extensive runtimes
- ... ... ut accurate results and good debugging possibilities

**Diagram:**

- **Input trace**
- **Model**
  - Application hardware platform mapping
- **Output trace**
Example  ace- ased Simulation

- **A stract simulation at system-le el it out timing**
  - aster than simulation but still based on a single input trace

- **A straction**
  - pplication - represented by *abstract execution traces* \( \rightarrow \) *graph of events: read, write, and execute*
  - chitecture - represented by “virtual machines” and “virtual channels” including non-functional properties (timing, power, energy)

- **teps**
  - xecution trace determined by functional application simulation
  - xtension of the event graph by non-functional properties
  - Simulation of the extended model

![Diagram showing the process of simulation and abstraction](image)