POLITECNICO DI MILANO

Development of Methodologies and Tools for Predicable, Real-time LEON-DSP based embedded systems

Marco Lattuada

Department of Electronics, Information and Bioengineering
Politecnico di Milano
marco.lattuada@polimi.it

December 11th, 2013

Outline

- Overview of the problem
- State of the Art
- The MPPB platform
- Design scenario
- Proposed methodology
- Implementation
- Predictability vs. Performance
- Conclusions

Heterogeneous Multiprocessor Platforms

- In many application fields, single core processors showed their limit in terms of processing power and system budget
- Heterogeneous multiprocessors platforms allow to better exploiting available resources to solve complex problems
- Examples of heterogeneous architectures:
  - General Purpose Processors + Dedicated Graphical Units
  - System on chips for mobile devices
- Porting of legacy C code applications designed for single core system to new heterogeneous multiprocessors systems can be an hard task
  - (Semi-)Automatic toolchains are required

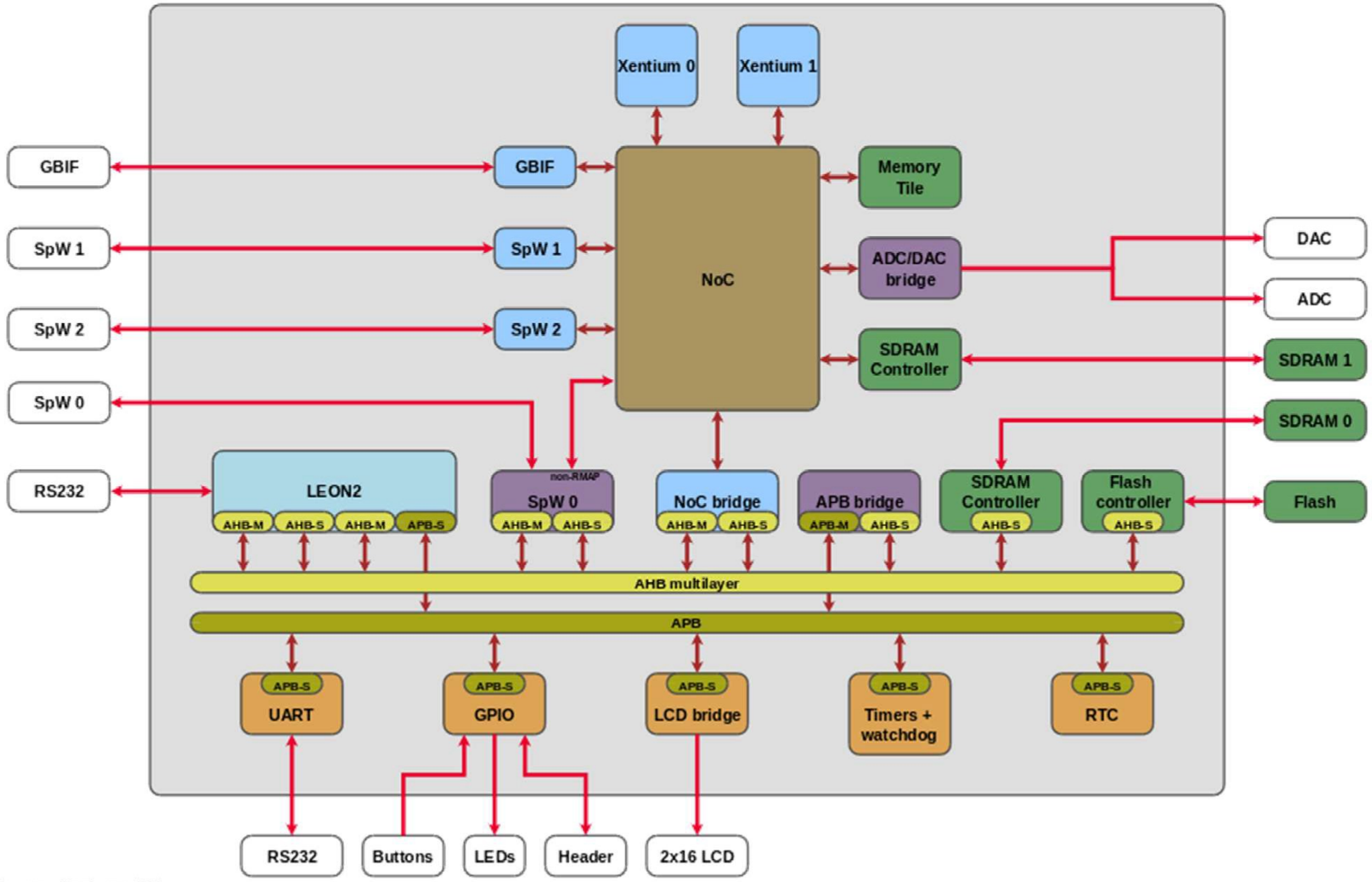
Heterogeneous Multiprocessor Platforms for Space Systems

- Characteristics derived from generic Embedded Systems:
  - Types of processing elements: General Purpose Processors (GPP), Digital Signal Processors (DSP) and Field Programmable Gate Array (FPGA)
  - Usually shared memory
  - Lightweight or no operating system, so reduced overhead due to thread management
    - Applications can be decomposed in smaller pieces
- Characteristics of Space Systems:
  - Available resources can be very limited
  - Predictability analysis must be guaranteed

Problem addressed by this NPI research

- Formulation and implementation of methodologies to (semi)-automatically port a sequential C code to a heterogeneous platform for space system
- NPI research project in collaboration with POLITECNICO DI MILANO, RECORE
- Input:
  - Legacy C source code applications
  - Annotations provided by designer
- Output:
  - C Parallel application targeting heterogeneous platform
    - Massively Parallel Processor Breadboarding (MPPB), developed by RECORE
- Solution: C to C compiler

MPPB Platform: Overview



Recore Systems BV

MPPB Platform: Characteristics

- Components of the architecture:
  - LEON2 processor
  - Two XENTIUM DSPs
  - Heterogeneous memories
  - High speed interfaces
  - Network on Chip + Bus
- No operating system: “Bare Metal”
- NUMA Distributed Shared Memory platform
  - Shared memory: common address space
  - Distributed: several memory devices
  - NUMA: Non-Uniform Memory Access:
    - Different access time according to memory location

MPPB Platform: Memory Details

- LEON2:
  - I-Cache: 16KB – 2way set-associative – LRU policy
  - D-cache: 16KB – 2way set-associative – LRU policy
- XENTIUM:
  - I-Cache: 8KB
  - Local Data Memory: 32KB
- Memories:
  - 256KB of SRAM close to XENTIUMs
  - 256MB of DDR connected to NOC
  - 256MB of DDR connected to bus
    - 128MB cached → private memory of the LEON2
    - 128MB non-cached → shared memory

Design Scenario

- No Operating System
- Single Application
  - "*Single thread*"
  - Multiple tasks: different tasks run at the same time on the different processing elements
- All managed by application code
  - transfers XENTIUM object code
  - transfers data to/from XENTIUM
  - starts/waits for XENTIUM tasks
- DMA transfers exploited when possible
- Use of interrupts to signal end of
  - DMA transfers
  - XENTIUM computations
- I/O peripherals are not used

Problem addressed by this NPI research: Details

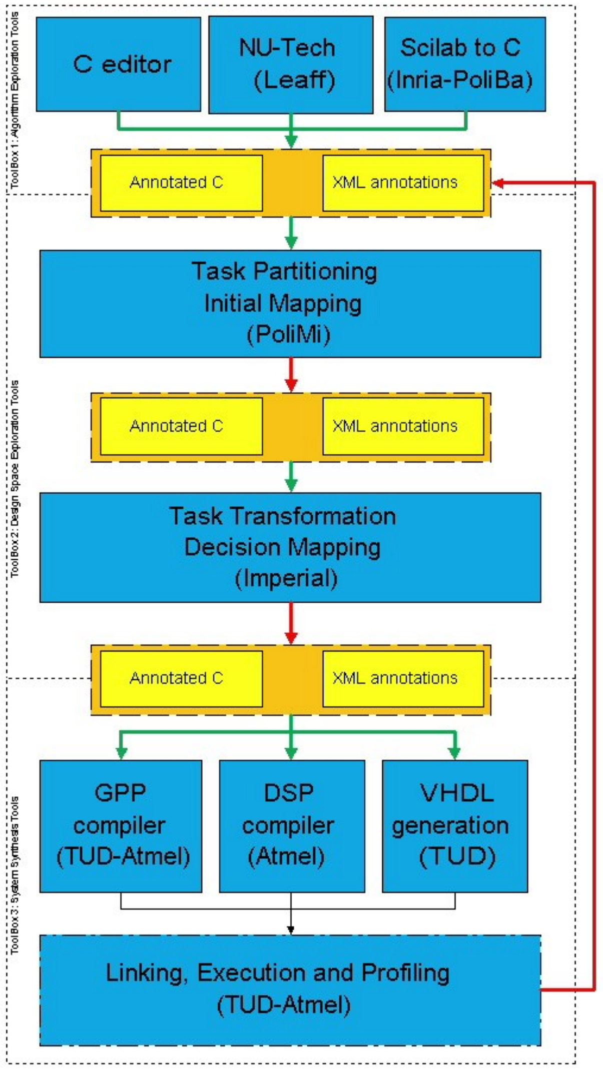
- Porting of annotated sequential C source code application to the MPPB platform
  1. Decompose the application in tasks
  2. Assign different tasks to different processing element
  3. Generates the C source code of the application

- Initial research work
  - Only performance optimization is considered (no power)
  - Complex data memory allocation and splitting are not addressed

Starting Point: the hArtes project

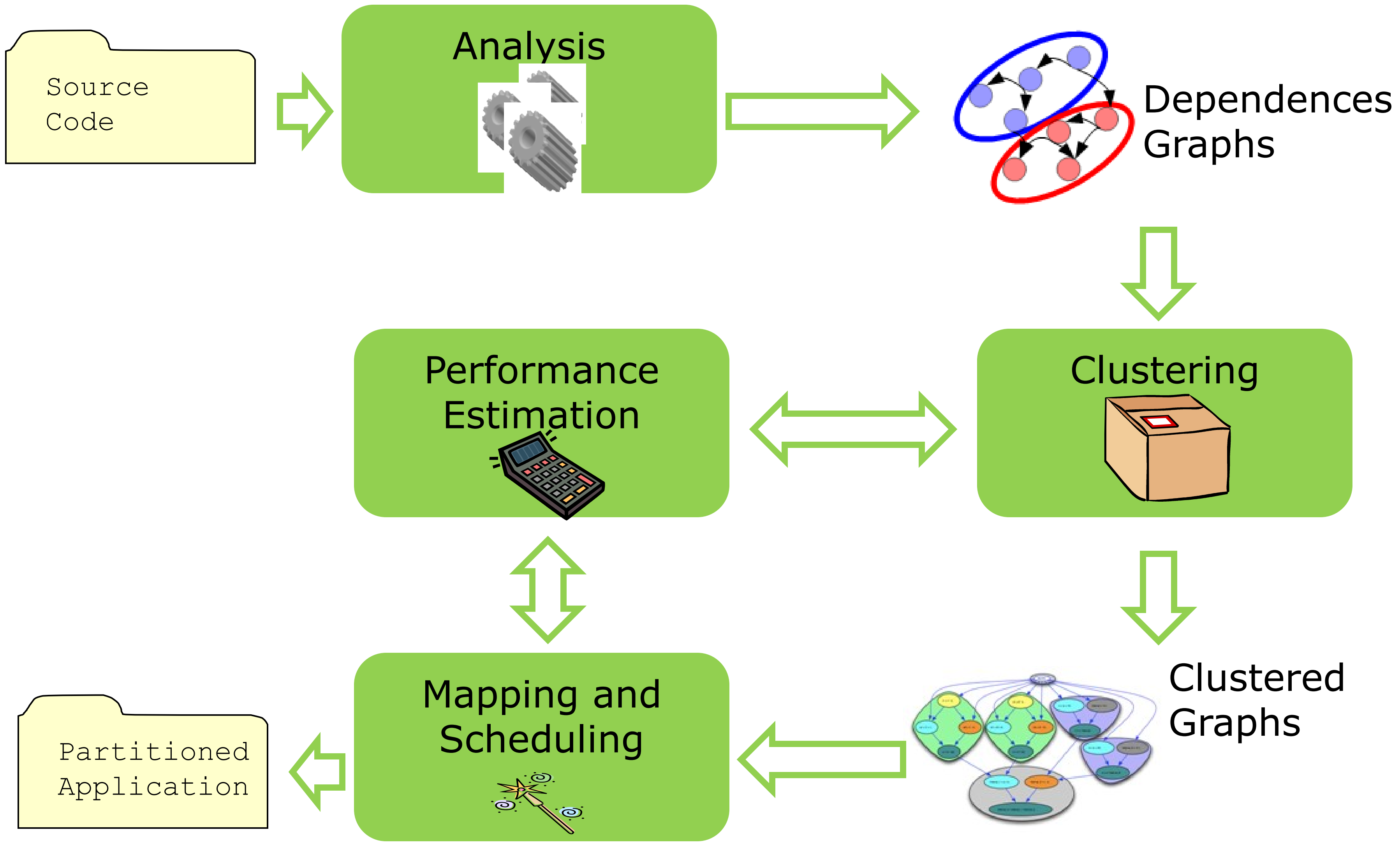
- **H**olistic **A**pproach to **R**econfigurable real **T**ime **E**mbedded **S**ystems (European FP6-IST project)
- Large project: 16 Industrial and Academic Partners – 17.34M euros budget
- Aim of the project is providing a new approach for designing complex and heterogeneous embedded solution.
- Two main contributions:
  - an heterogeneous multiprocessor platform composed by:
    - Atmel Diopsis D940HF (ARM + DSP Magic)
    - Xilinx XC4VFC140
  - A toolchain to easily exploit the computational power provided by the board

The hArtes Toolchain


ToolBox 1: Algorithm Exploration Tools
C editor
NU-Tech (Leaff)
Scilab to C (Inria-PoliBa)
Annotated C
XML annotations
ToolBox 2: Design Space Exploration Tools
Task Partitioning Initial Mapping (PoliMi)
Annotated C
XML annotations
Task Transformation Decision Mapping (Imperial)
Annotated C
XML annotations
ToolBox 3: System Synthesis Tools
GPP compiler (TUD-Atmel)
DSP compiler (Atmel)
VHDL generation (TUD)
Linking, Execution and Profiling (TUD-Atmel)

- Tools communicate through XML files and Source Code Annotations
- A subset of the OpenMP pragmas is used to describe the parallelism
- Profiling and mapping information expressed using custom pragmas
- Interactions with external tools introduce further constraints which have to be taken into account by proposed methodology

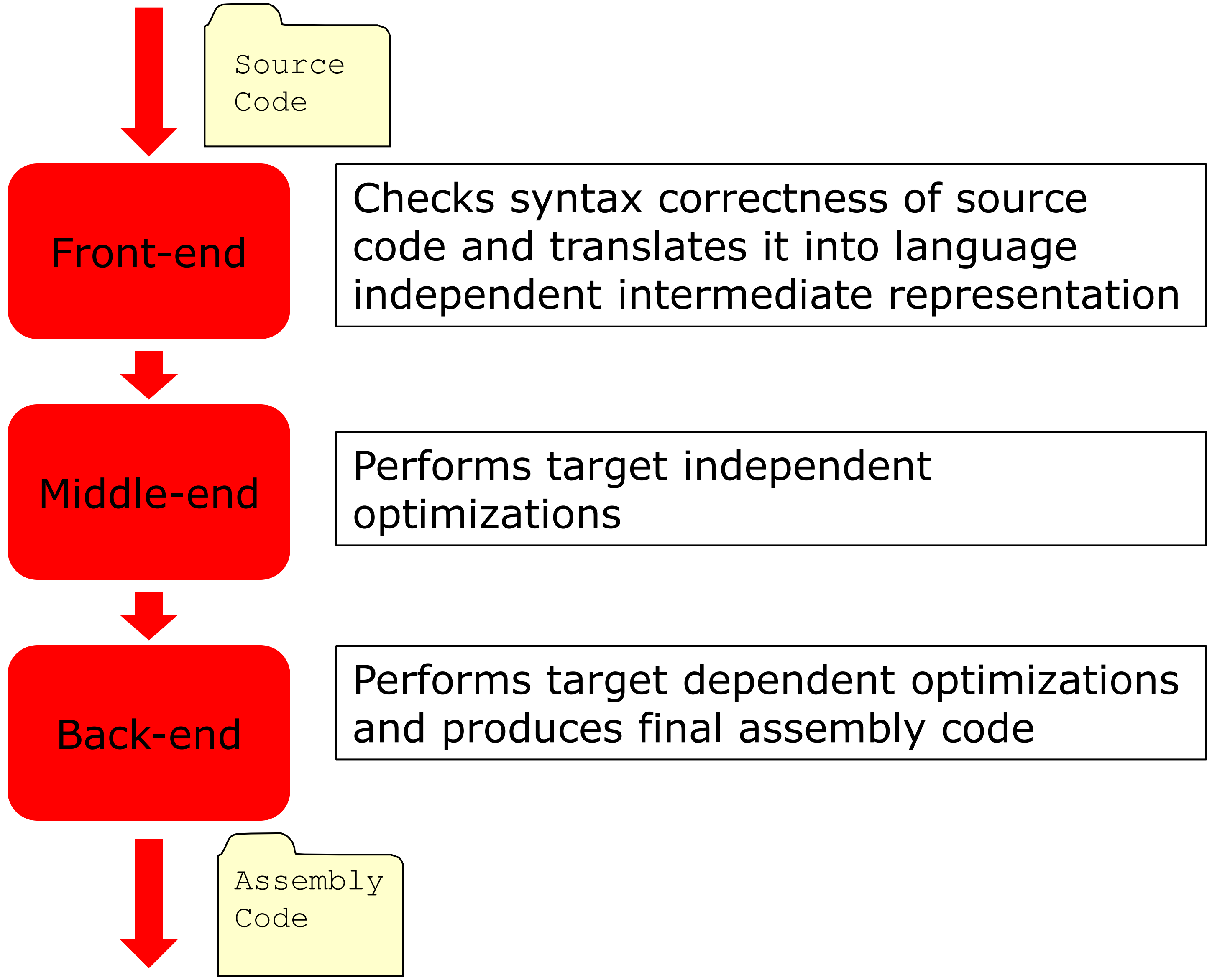
The Proposed Design Flow



Design Flow: Input

- Input of the design flow is the C source code of the application
- Complete application allows to produce better results
  - Target libraries can be exploited for profiling
- Designer can provide hints to the toolchain
  - By annotating source code specifying parallel portions of code or partial mapping solution
  - By limiting the analysis of the toolchain to part of the application
- Suggestions will be evaluated, but can be rejected if not profitable

Compiler structure

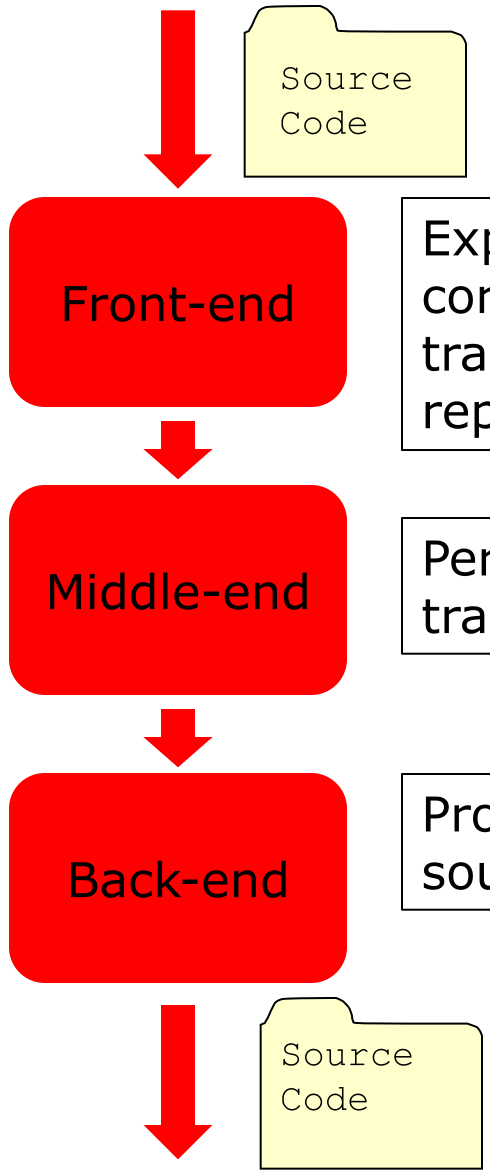


Checks syntax correctness of source code and translates it into language independent intermediate representation

Performs target independent optimizations

Performs target dependent optimizations and produces final assembly code

Design Flow: Compiler-Like Structure



Exploits GNU GCC to checks syntax correctness of source code and translates it into intermediate representations

Performs optimizations and transformations

Produces the partitioned application source code

Design Flow: Analysis of Source Code

- Internal Representations of GCC extracted by means of a GCC plugin are the starting point of the analysis
- All the state-of-the-art analyses and optimizations are performed by GCC
- Performance estimation can produce inaccurate results if based only on static information
- Dynamic information such as number of loop iterations or branch probabilities can improve quality of results
  - Source code of the application is instrumented and sequentially executed on the target or on the host architecture to collect this information (path profiling)

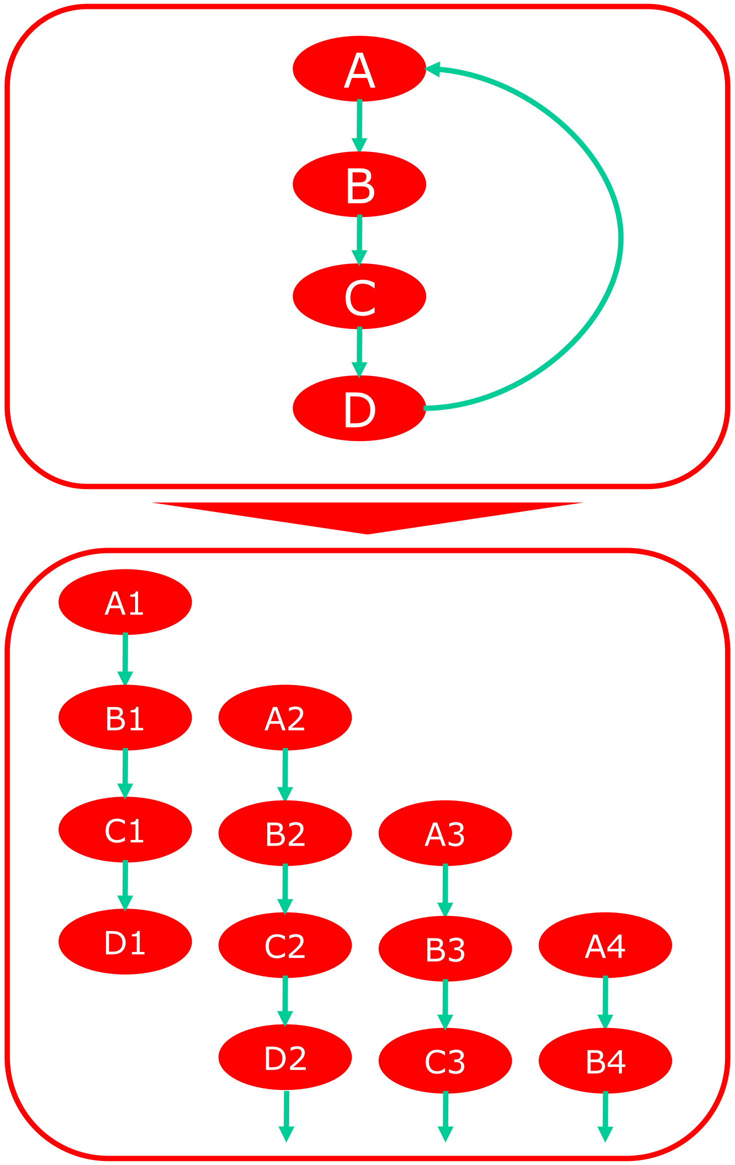
Design Flow: Dependence Analysis

- Identify all the dependences (control + data) between operations
- One of the most critical parts
  - It has to be conservative: ignoring real dependences can produce wrong code
  - It has to be accurate: add false dependences can reduce the parallelism of the application
- Three approaches:
  - GCC alias analysis
  - GCC alias analysis + refinement
  - Dynamic Data Dependence analysis
    - must be validated by the designer

Design Flow: Partitioning

- Extraction of clusters of operations from sequential application
- Based on analysis of dependences graphs
- Three different strategies
  - Sequential grouping or cutting
  - Parallelism extraction
  - Pipeline extraction
- Produces Hierarchical Task Graphs and Synchronous Data Flow Graphs:
  - Potentially a graph for each loop of each function plus a graph for each whole function
  - Nested loop represented as single node in parent loop graph

Design Flow: Pipeline Extraction



Design Flow: Partitioning

- Not all the extracted parallelism can be actually exploited
- Real platform has to be taken into account
  - Limited number of processing elements: 3
  - Overheads due to data transfers and task synchronizations
  - Limited available memory
- Fast estimation used to evaluate intermediate solutions and to dimension tasks

Design Flow: Mapping and Scheduling

- Each task has to be assigned to a processing element
- Implied data transfers have to be assigned to communication elements
- Execution of order of task and communications assigned to the same processing element is statically computed
- Elaboration and communication can be parallelized
- If a tasks is assigned to a DSP, all the functions called by it are assigned to the same DSP
  - Potentially reduces the number of possible solutions
  - However a function can be assigned to more than a processing element depending on its call points

Design Flow: Mapping and Scheduling

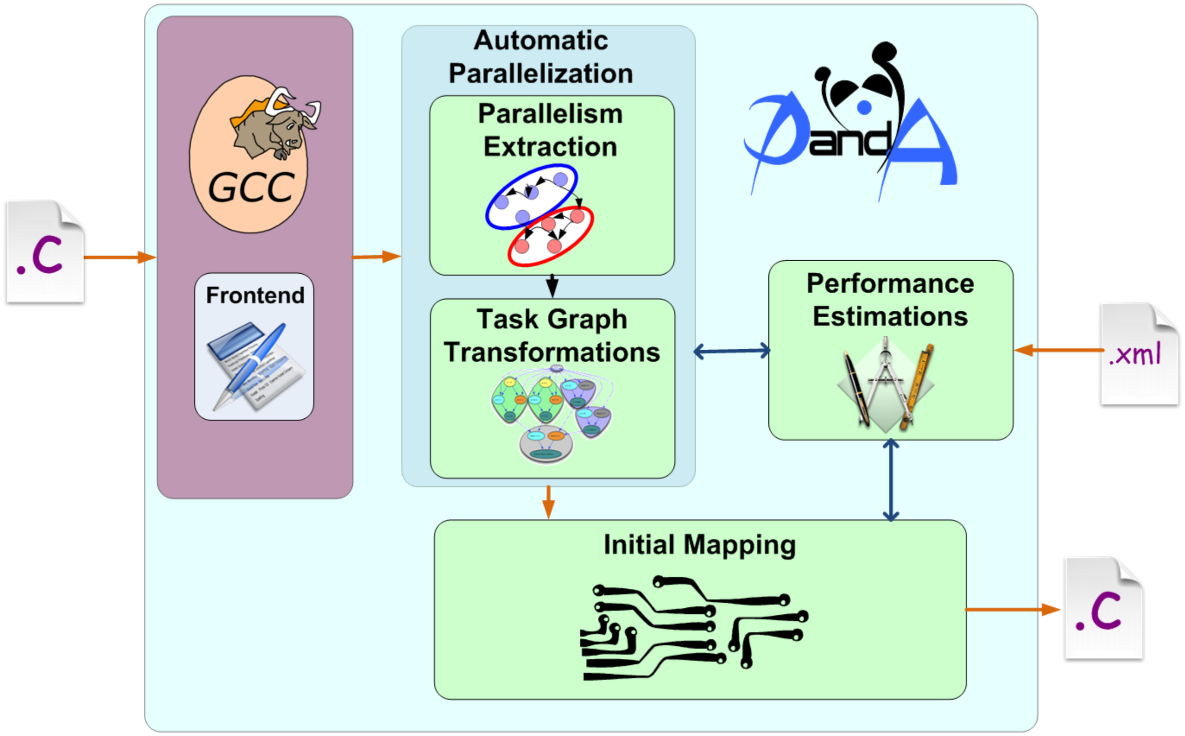
- Assigning each task to the fastest processing element can be not the best solution:
  - Overhead due to communication
  - Delay of tasks assigned to the same processing element
- Different possible design solutions:
  - Ant Colony Optimization heuristic adopted to compute the final solution
  - Performance estimation exploited to evaluate the different solutions

Design Flow: Backend

- Selected solution is written back in form of C source code
- Data have to be explicitly transferred so their size must be known
  - It can be a critical aspect in presence of pointers
- Target is space system, so rules to allow predictability analysis of the code must be followed
  - e.g. not introduce further dynamic allocation
- At the moment MPPB can be exploited only through MPPB API
  - TASTE infrastructure does not yet support this programming model nor MPPB platform

Implementation

- The proposed methodology is being implemented in Zebu, a tool part of the PandA framework
  - hw/sw co-design framework based on GNU/GCC compiler



http://panda.dei.polimi.it/

Case Study

- Algorithm to process raw frames coming from the near infrared (NIR) HAWAII-2RG detector
- Open source application provided by ESA for benchmarking purpose; frames data size can be customized
- Frame data size has been limited to fit frames in XENTIUM local data memory
  - Optimization of data allocation and splitting is out of the scope of this research work
- Code slightly modified and annotated
  - Random input data have been embedded in the application instead of generated at run time
- Pipeline parallelism identified: up to 7 stages
  - Reduced to 3
- Speed-up obtained (w.r.t. LEON2) 1.56x

Predictability vs. Performance
Single Application

- Source of uncertainty:

1. LEON Caches
   - Same situation of single core platform
2. XENTIUM Instruction Cache
   - Ideally XENTIUM code fits in the Instruction Cache
3. Interrupts
   - Some of them can be replaced with busy-waiting potentially decreasing the performance of the application
4. Network on chip communications
   - Can be sequentialized

- Non-Source of uncertainty

1. XENTIUM Data Memory
   - Completely controlled by application
2. Scheduling
   - Completely Static

Predictability vs. Performance
Multiple Applications

- Operating system without explicit support to MPPB
  - Applications running only on LEON does not require special treatment
  - Applications offloading tasks to XENTIUMs cannot be preempted during the whole offload
    - This can potentially reduce resource utilization
  - Multiple communications can introduce further uncertainty
- Operating system with explicit support to MPPB
  - Significant modifications are required:
    - Data transfer management
    - Processing element – task affinity
  - No hardware support for task switching on the XENTIUM
  - Schedulability analysis has to take into account further constraints

Conclusions

- A design flow for porting sequential application to heterogeneous multiprocessors embedded systems has been presented
- The proposed flow integrates:
  - Application analysis based on exploitation of GNU GCC intermediate representation
  - Application partitioning based on vertices clustering
  - Performance estimation using performance models built with linear regression
  - Mapping and scheduling based on Ant Colony Optimization
- The proposed flow generates source code exploiting MPPB API