PACHA

A low overhead, Platform Agnostic, Close-to-HArdware programming interface

Alexandre Aminot, Alexandre Guerre, Yves Lhuillier

Contact: alexandre.guerre@cea.fr

CEA, LIST, Embedded Computing Laboratory,
F-91191 Gif-sur-Yvette, France
Outline

1 - Context

2 - PACHA

3 - Results
1 - Context
Context

Shared-Memory Multi-core

Different Hardware Architectures & Programming Models
Exploring **early** in the design flow the triplet

\{ Algorithms, Parallel programming models, Architectures \}

- Apps
- Knowledge Database
- Empirical Database
- Automatic tools
- Formal methods
- Choices: Architectures, Parallelization methods

**Context: CoTs Project**
How to run our applications on different many-core architectures?

Linux, VM: Not adapted to embedded system

OpenACC, OpenCL ...: not supporting the targeted architectures, yet

A bare metal version for each architecture: High development cost, objectivity of the comparison?

- No common tools for bare metal environments
Context: Bare metal

Bare metal has **Less overhead** than Linux

**But**

- Hard to debug
- Proprietary API: Long learning curve
- No common parallel execution models
2 - PACHA
Abstract the complexity of Bare metal Environments

Programming Interface

Development Environment
Software stack (C + ASM)

Applications
1 unique version
Running on multiple Architectures

Libs & API
Parallel Models
Unique Base API

Archi
1 or more
Backend by archi

PACHA API
PACHA Libs
Applications
Architectures

Architectures

Architectures

Architectures
PACHA API

The minimal set of function

Lock_acquire(lock *)

Generic Types

lock, coreset, procid ...

Generic Execution context

Entry Point(), init() and destroy()

Static Inline for the API Functions

+ Constant values and entry point with #define

Modules

Lock
Memory
Platform Management
Performance
Task
Trace
PACHA: Development Environment

(1) **Platform independent** parallel execution Models

(2) Abstract proprietary environment

Diagram:
- Application
  - Makefile
  - Makefile.common
  - Makefile.platform
- PACHA
  - Makefile.platform
PACHA: Development Environment

(3) Linux functional simulator

Cores are abstract by linux

Develop and debug on your own workstation
PACHA: Add easily a backend

Implement the development environment
  Compilation, deployment rules

Implement the API
  ~ 600 C and 15 ASM Lines of Code
  Request: memory manager and lock manager
  -> functional tests available

Fill the documentation layer
  Platform option, Implementation details
3 - Results
Results: Current state of the tool

4 Platforms

Tilera Tile64 - Bare metal
  - Linux

ST microelectronics/CEA Sthorm – Bare metal (simulator)

Locomotive SystemC simulator

X86_64 - linux
Pedestrian Detection

Preprocessing
Static part
Coarse grain

Classification
Dynamic Part
Fine Grain
64 independent plans

Results: Application overview
Results: Tilera, Baremetal vs Linux

Pedestrian Detection with 2 different back-ends

Preprocessing

Classification

Tilera Baremetal (BME) 4x faster than Tilera Linux
Results: Parallel execution model

PACHA vs common parallel execution models

Preprocessing

Classification

PACHA Baremetal 2x faster than OpenMP and Pthread
Simple study case example:
Imagine you as an embedded system designer (or be yourself),
Which platform are you likely to use with a workload like the Pedestrian Detection application?
Results: Architecture comparison

X86 (AMD buldozer) or Tilera sound interesting
Results: Architecture comparison

considering power efficiency, Sthorm or Tilera sound more interesting
Conclusion

Bare Metal environment: efficient access to dedicated hardware but high cost development

PACHA: Unified API + deployment environment

- Multiprocessor comparison for the same target application.
- Efficient access to bare metal environment
- Functional Simulator + Debugging tools
- Unified parallel execution models
Thank you,
Any question?