CAPS

Innovative tools for a new paradigm

Programming Manycore Embedded High Performance Applications

Embedded Software, December 16th, 2008

Introduction

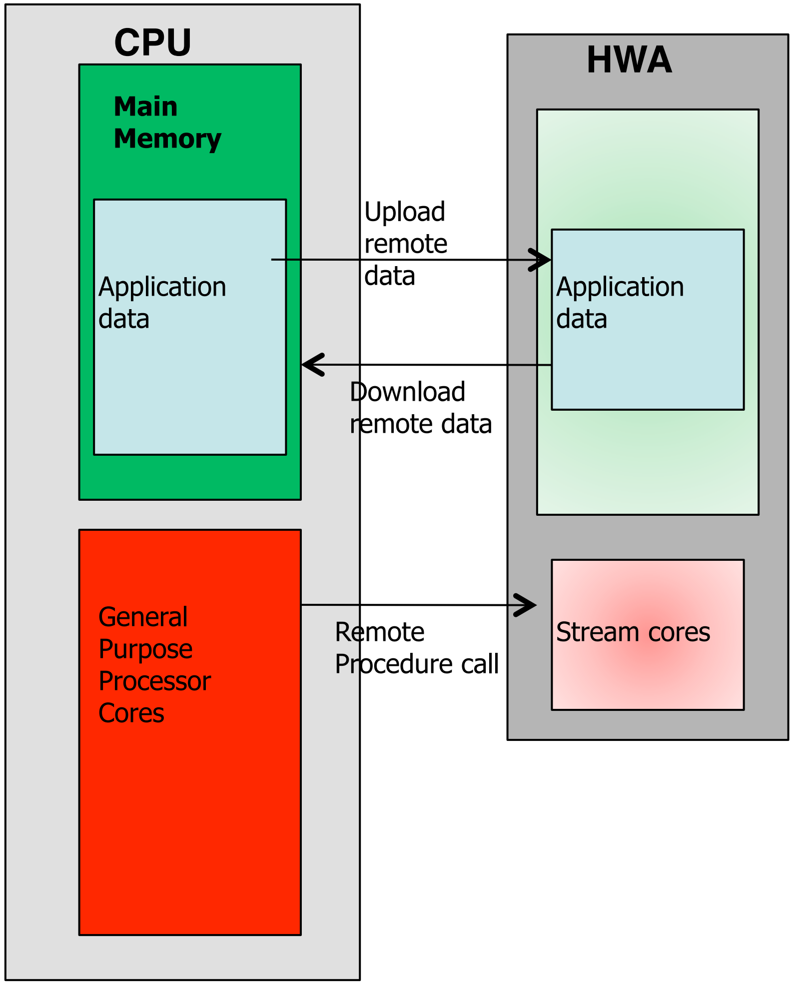
- High performance embedded applications rely on new multicore architectures
  - It is about performance not parallelism
- Various hardware
  - General purpose multicores
  - Application specific (DSP)/configurable processors
- Moore's law still applies
  - Doubling number of cores every ~18 months
  - Operation/Watt is the efficiency scale
- HPC and embedded applications are increasingly sharing characteristics

Overview

- Manycore architectures
- Challenges
- Compilers for embedded manycore architectures
- Milepost project
- The Multicore Association
- Conclusion

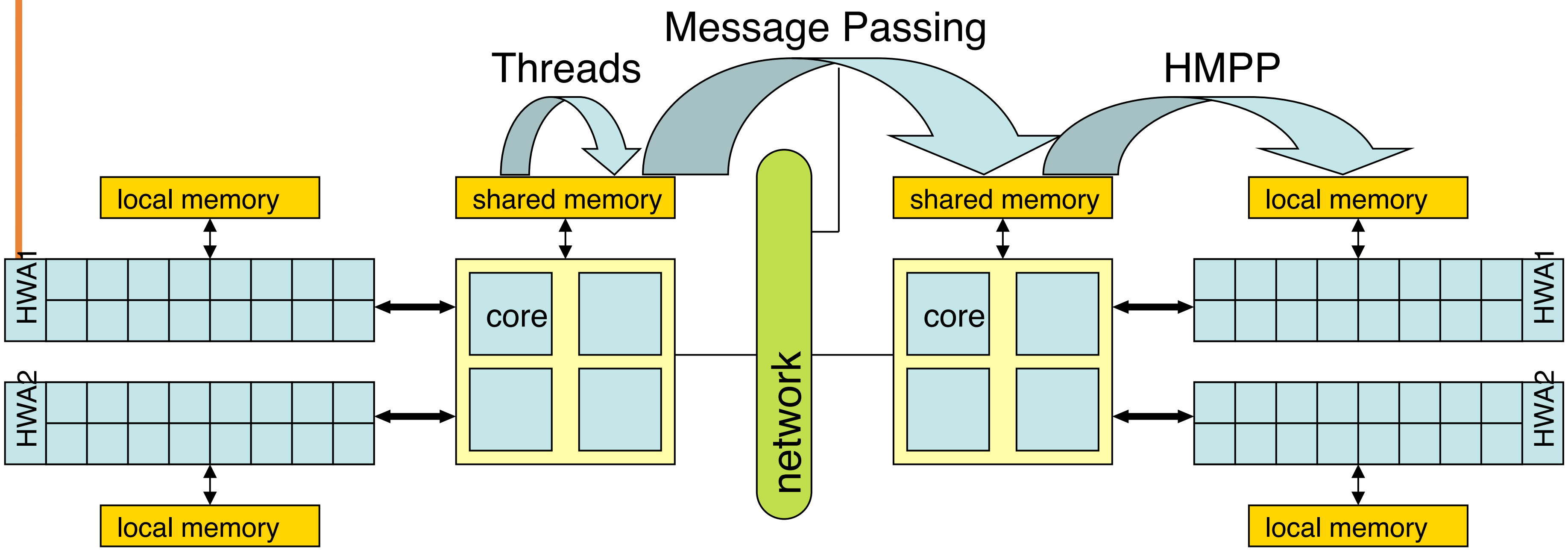
Manycore Architectures

- General purpose cores
  - Share a main memory
  - Core ISA provides fast SIMD instructions
- Streaming engines / DSP / FPGA
  - Application specific architectures ("*narrow band*")
  - Vector/SIMD
  - Can be extremely fast
- Hundreds of cumulated GigaOps
  - But not easy to take advantage of
  - One platform type cannot satisfy everyone
- Tilera, TMS320TCI6488, Cell, …



Multiple Parallelism Levels

- Amdahl's law is forever, all levels of parallelism need to be exploited
  - Hybrid parallelism needed

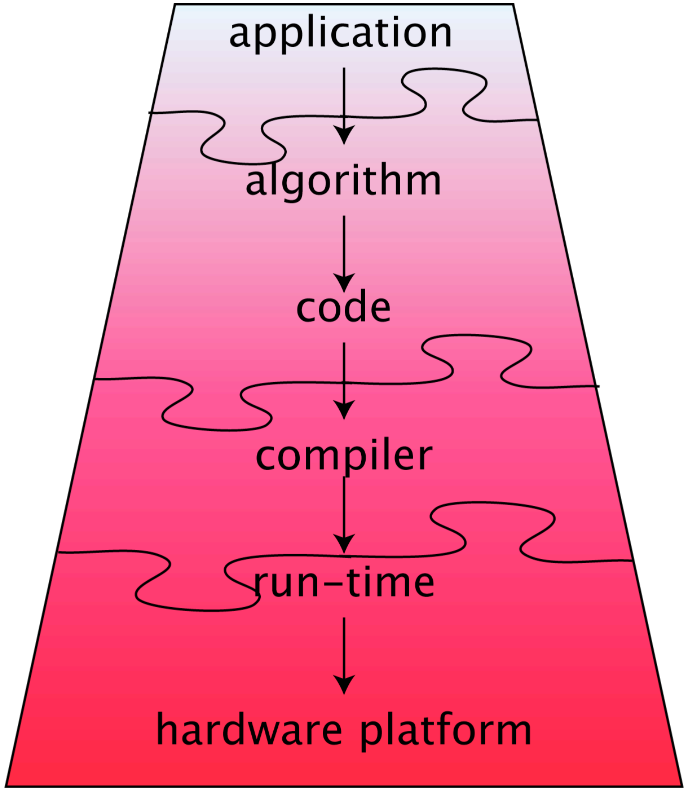


The Past of Parallel Computing, the Future of Manycore?

- The Past
  - Hundreds of parallel languages were proposed
  - Scientific computing focused
  - Microprocessor or vector based, homogeneous architectures
  - Trained programmers
- The Future
  - New applications (multimedia, medical, …)
  - Thousands of heterogeneous systems configurations
  - Asymmetry issue

The Challenges

- Programming
  - Medium
- Resources management
  - Medium
- Application deployment
  - Hard
- Portable performance
  - Extremely hard



What is Specific to Embedded App.?

- Co-design / co-configuration issues
- (Soft) Real time issues
- Need light weighted environments
- Short system lifetime
- Hardware may not exist

Research Directions

- New Languages
  - X10, Fortress, Chapel, PGAS languages, …
- Libraries
  - Atlas, MKL, Global Array, Spiral, Telescoping languages, TBB, …
- Compilers – Key for the short/mid term
  - Classical compiler flow needs to be revisited
  - Acknowledge lack of static performance model
  - Adaptative code generation
- Architectures
  - Integration on the chip of the accelerators
    - Fusion, …
  - Alleviate data transfers costs

Compiler Focus

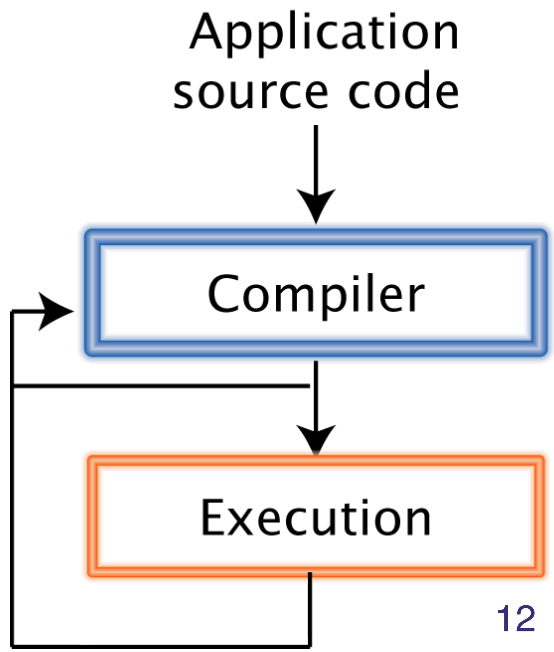
- Current compilers
  - Have insufficient understanding about program input, architecture
  - Have to deal with a very large optimization space
  - General purpose tools
- Compilers are not good at
  - Understanding whole programs
  - Understanding performance
  - Making decisions
    - Finding global optimization strategies
    - What code (suite of) transformations and when ?
- Compilers are good at
  - Dealing with local compute intensive tasks
  - Transforming, duplicating, specializing, generating codes

Future of Compilers for Manycores

- What's new!
  - More processing time can be spent on the code generation and optimization processes
- Mix offline and online techniques
  - Iterative compilation
  - Machine learning
  - Speculative techniques
  - Adaptation
  - Runtime compilation and optimization
  - Better understanding of libraries

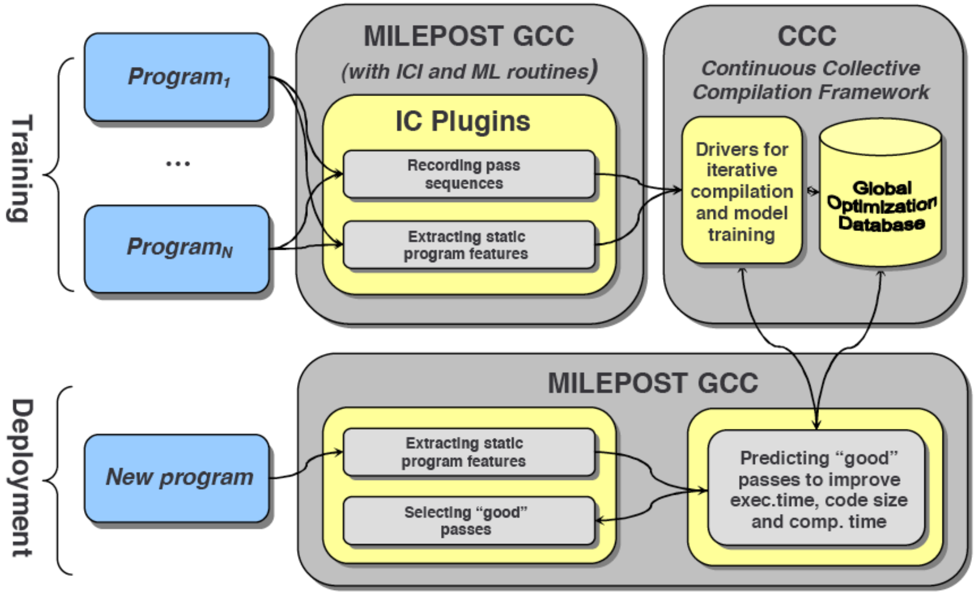
Iterative Compilation

- Use multiple compilations to select the best optimization strategy according to feedbacks
  - Static - Analysis of the output code according to a performance model
  - Dynamic - Performance measurement
- Pros
  - Explore the optimization space (for instance tiling block size)
  - Usually find better results than human
  - Cheap (when static)
- Cons
  - Expensive (when dynamic)
  - Complex compilation flow
- Some related works
  - Maqao, CapsTuner, Milepost, ACME, Autotuner, ESTO, Atlas, FFTW, … …



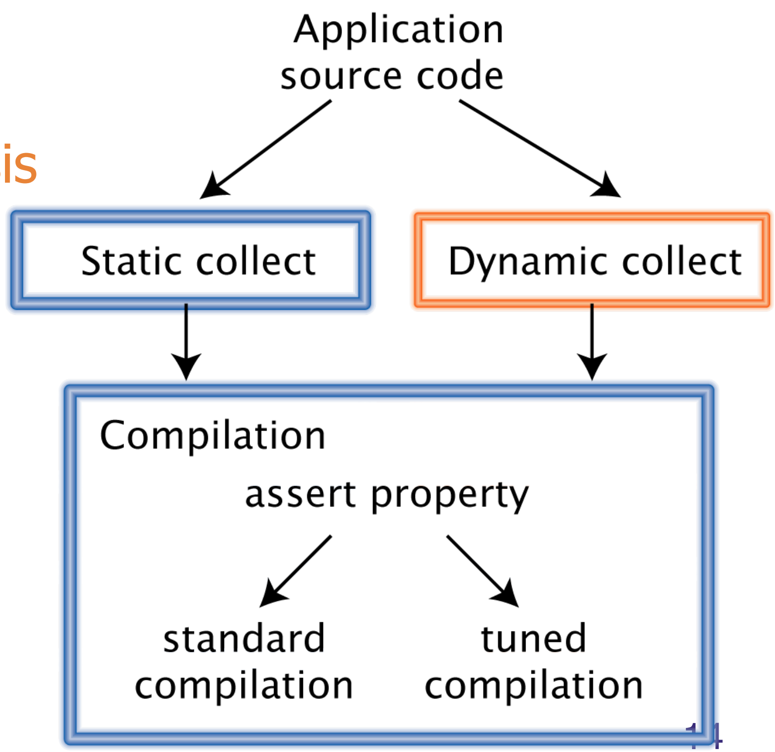
Machine Learning

- Learn from previous compilations and executions
  - Use static and dynamic features
  - Avoid iterative compilation
- Pros
  - Efficient, compilers easier to build
- Cons
  - Overfiting of the training set
  - Scope ?
- Some related works
  - GCC-ICI, Milepost, Meta Optimization, …



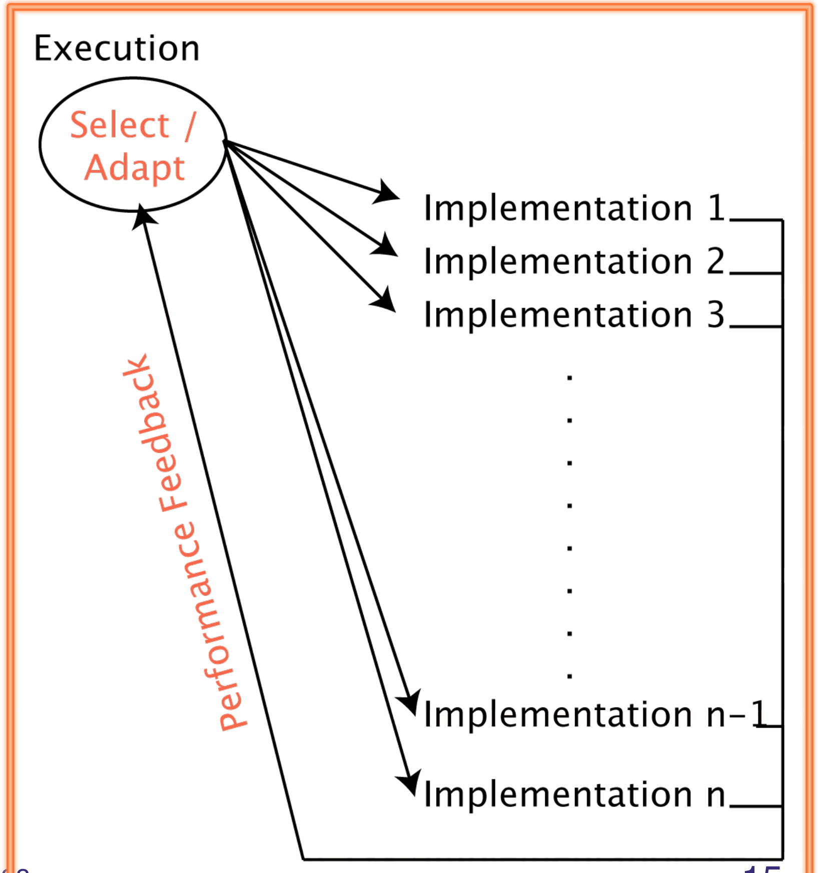
Speculative Techniques

- Assumes "a priori" properties of the code to achieve parallelization or optimization
  - Code specialization
  - Check at run-time if properties are true
- Pros
  - Allow better code optimizations
  - Help avoiding inter-procedural analysis
- Cons
  - Execution overheads
  - Overspecialization
- Some related works
  - Parasol, VESPA, Nemalabs, …
  - CAPS Codelet Finder



Adaptative Techniques

- Adapt to execution context while running
  - Measure performance and select implementation while running the code
- Pros
  - Take into account real efficiency
- Cons
  - Runtime or code overheads
  - Multi-path code acceptance
- Some related works
  - Stapl, Unidap, tbb, …



Runtime Optimization and Compilation

- Code generation/optimization according to execution context
  - Stream computing oriented, …
- Pros
  - Can deal with non existing hardware when packaging the application
  - Accurate/exhaustive context information
- Cons
  - High overhead
  - Limited scope (especially pure runtime or binary level)
  - Safety and debugging
- Some related works
  - RapidMind, Accelerator, (Dynamo,) …

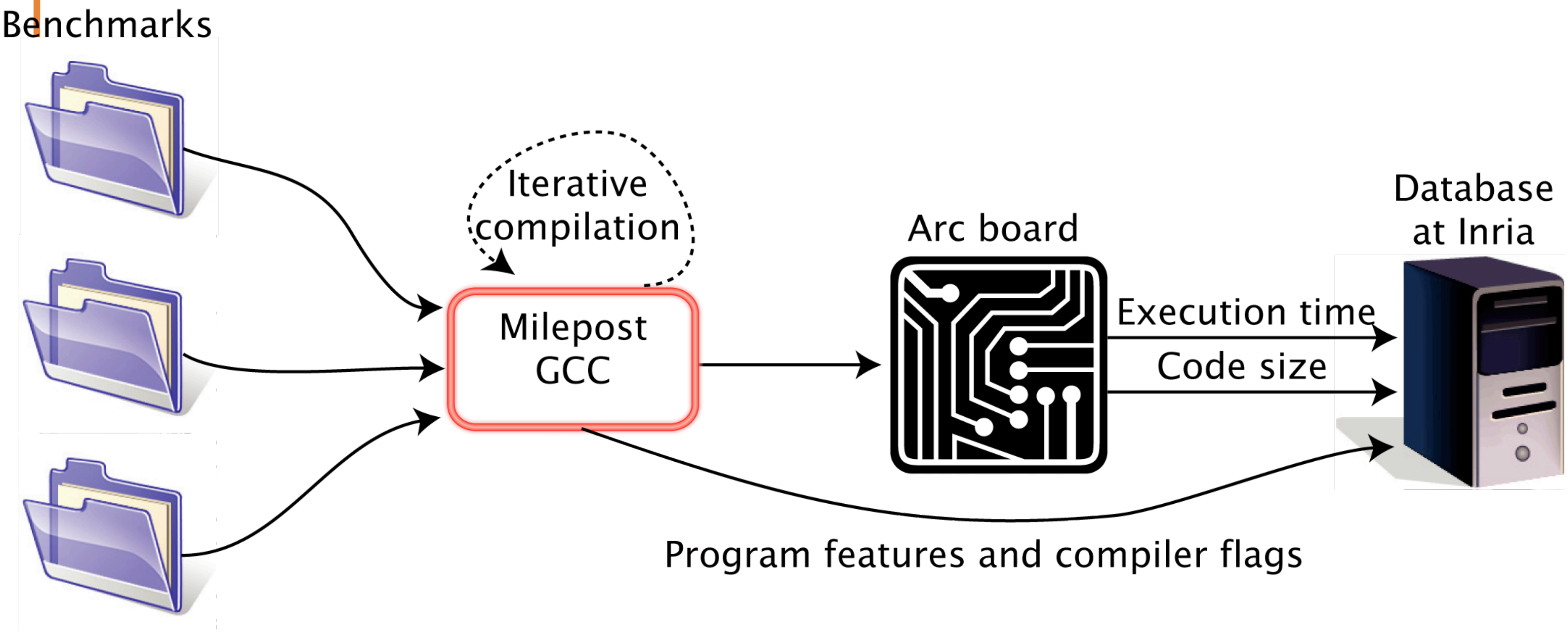
Milepost Compiler

- Objective
  - To develop compiler technology that can automatically learn how to best optimise programs for re-configurable heterogeneous embedded processors.
- Partners
  - University of Edinburgh, ARC International Limited, CAPS-Entreprise, IBM Israel - Science and Technology, INRIA

`http://www.milepost.eu/`

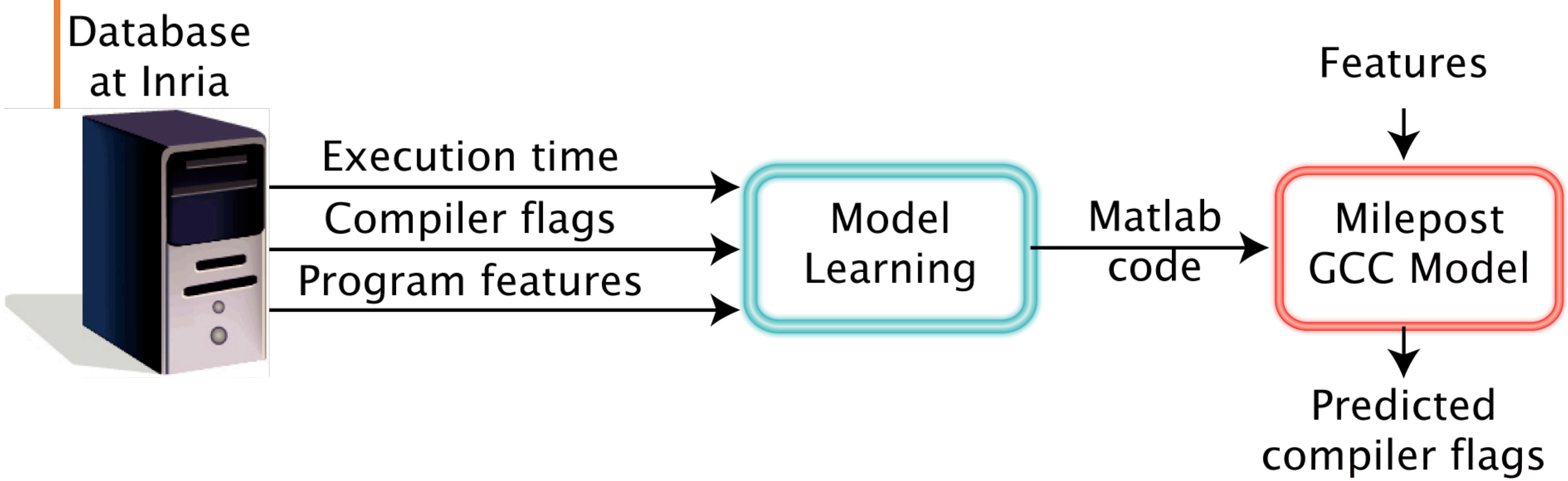
Milepost Overview - 1

- Database filling with a training set



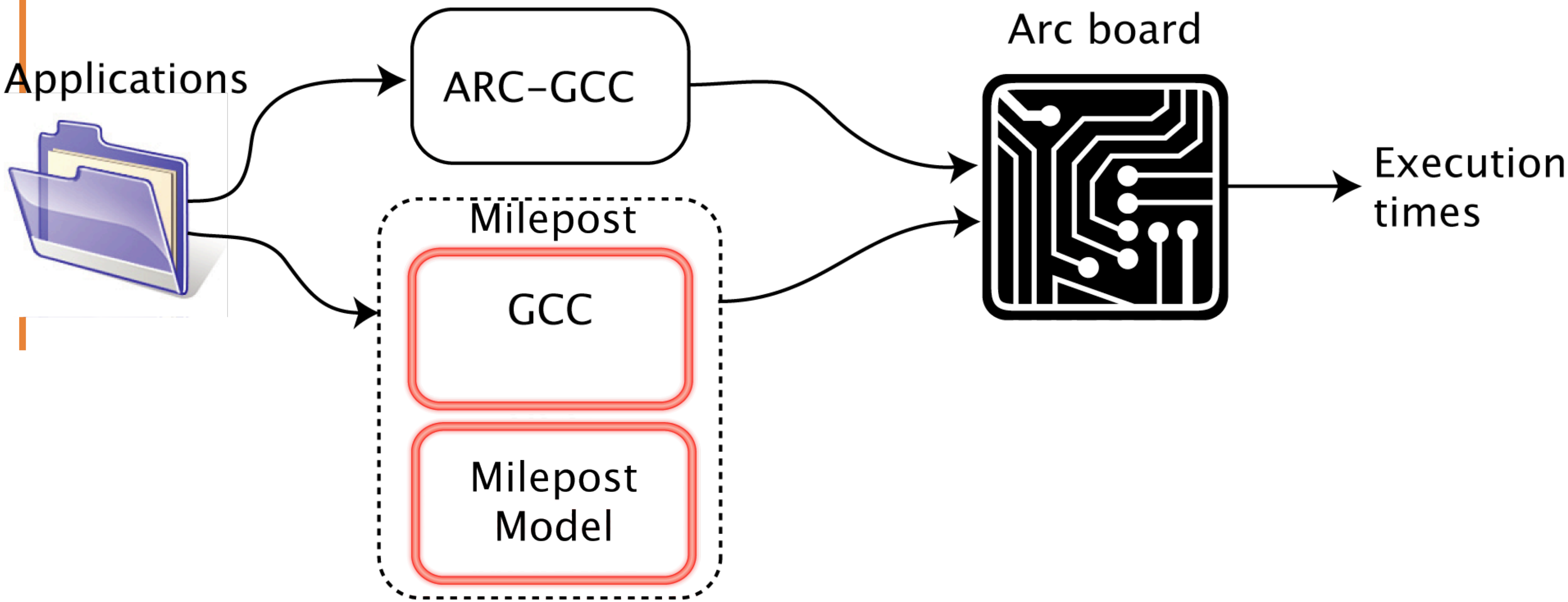
Milepost Overview - 2

- Building the Model



Milepost Overview - 3

- Using the improved compiler



Milepost Compiler Status

- Prototype is available
  - Provide an average of 11% performance improvement
- More details
  - `http://gcc-ici.sourceforge.net/papers/fmtp2008.pdf`

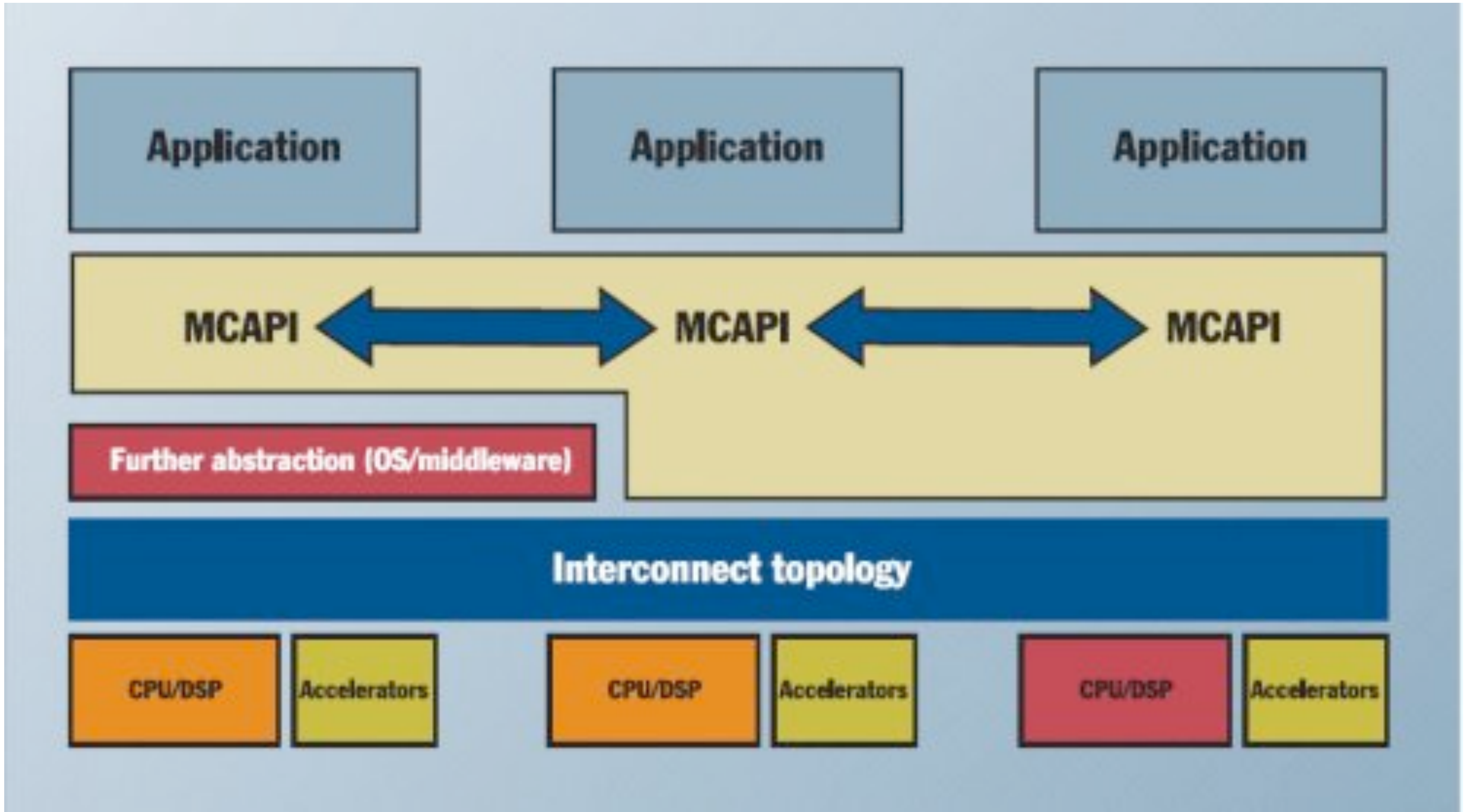
Multicore Association (MCA)

- MCA is an open membership organization about multicore technology
- Working groups
  - Communications API
  - Programming Practices
  - Resource Management API
- Members
  - CAPS entreprise, Codeplay, CriticalBlue, IMEC, Freescale, Intel, TI, Tilera, Virtutech, Wind River, …

http://www.multicore-association.org/

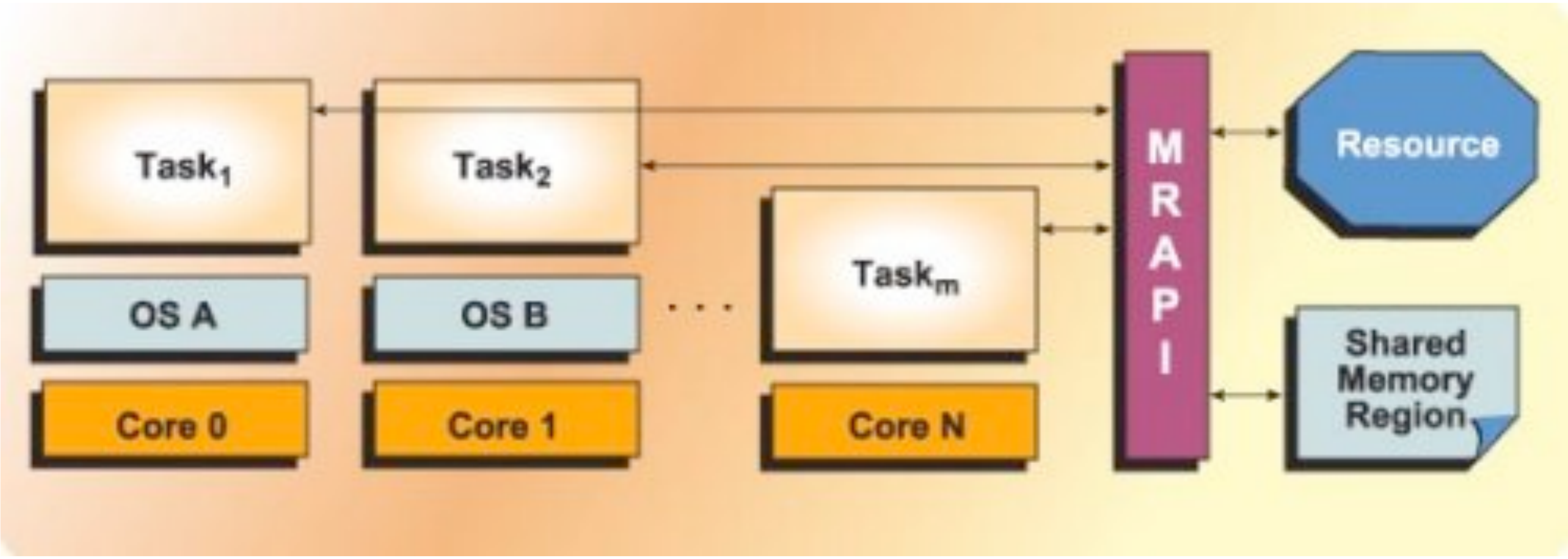
Communications API (MCAPI)

- MCAPI is a message-passing API



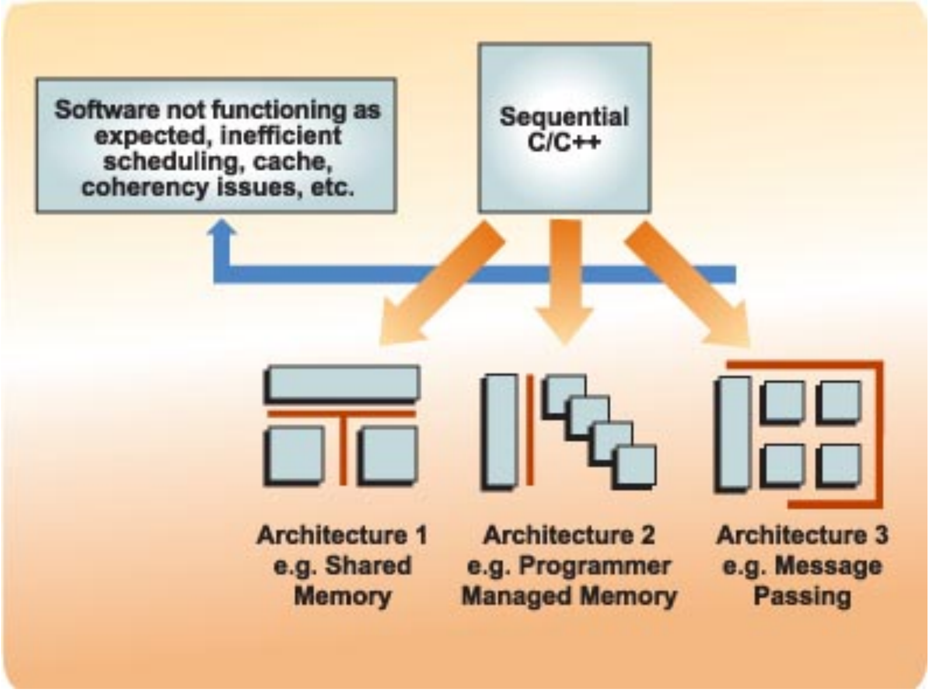
Resource Management API

- Defines an industry-standard API that specifies essential application-level resource management capabilities



Programming Practices

- Objective
  - To define industry-wide, best practices to leverage existing code in multicore environments
- How today's C/C++ code may be written to be "multicore ready"



Conclusion

- Very exciting time for compilers!
  - We need to understand how much CPU time should be used for discovering/managing parallelism
- But should not be in charge of dealing with coarse/large grain parallelism
  - Node/socket level issues only
- Next generation compilers should
  - behave “linearly” (i.e. have less threshold effects)
  - better interact with human
  - exploit application specific knowledge
  - generate very efficient sequential codes
  - deal with heterogeneous instruction sets
  - exploit stream/vector computing
  - deal with memory and computing resources allocation
  - deal with some fault issues
  - interface programs with power management