Will the software Dinosaurs step aside or step on MPSoC?

Martijn de Lange

ACE Associated Compiler Experts

Founder / CEO

MPSoC, July 2004

ACE Associated Compiler Experts

- Subsidiary of ACE Associated Computer Experts
  - Established 1975, Amsterdam, the Netherlands
  - Employee-owned company
  - Advanced systems-software products and services
- Products
  - CoSy compiler-development system
  - CoSy Express technology
  - SuperTest C compiler test & validation suite
- Target Market
  - Compiler developers
    - Semiconductor companies
    - Software tool vendors
- www.ace.nl

Overview - what’s a design

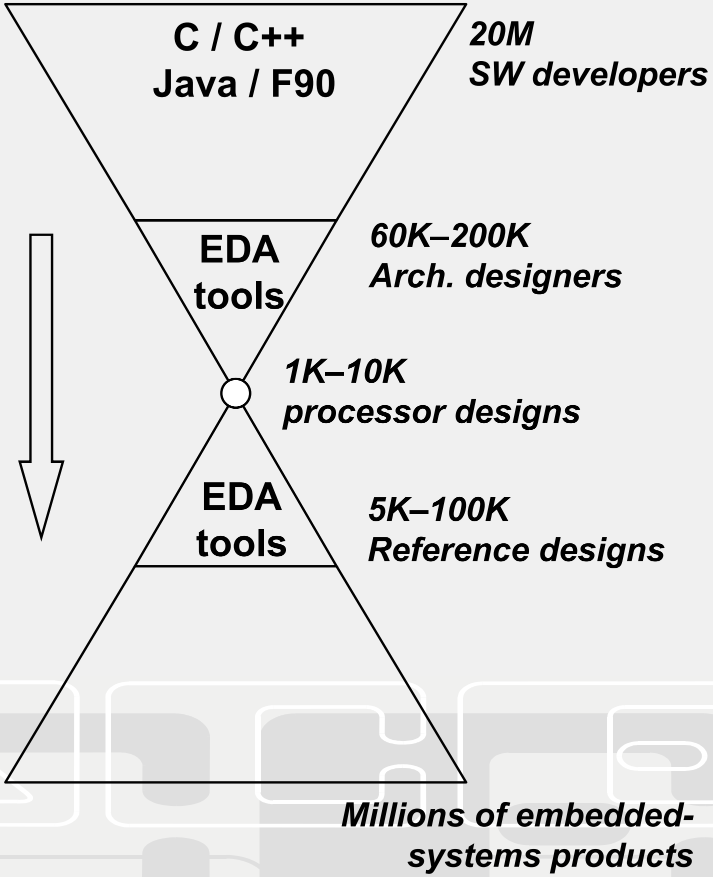
- MPSoC design moves too fast
- Software is dominant but not recognized
- ‘Design’ needs redefining
- H/W and S/W need teaming up
- Parallelism, why should we

Too Many Embedded Challenges

- Multi Processor, heterogeneity, legacy codes
- Design flexibility
- Programmers who won’t follow
- Software Costs > 50% of Design Cost
- Flexibility not utilized

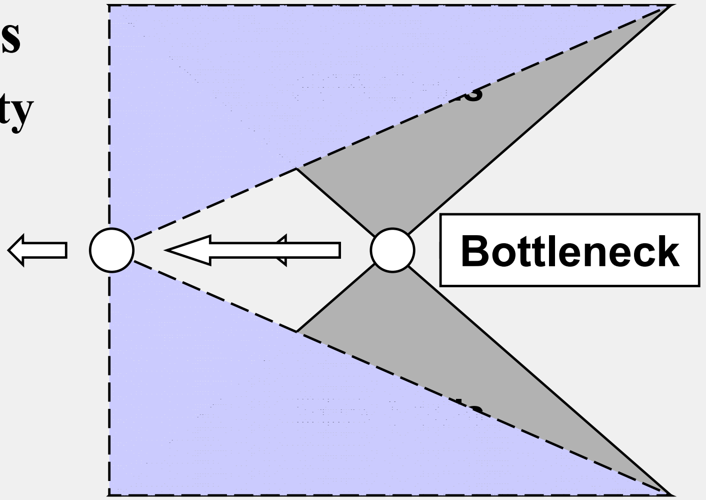
Embedded Systems Development Realm

- 20M SW developers
  - C, C++, Java, Fortran-90…
- 60K–200K arch. designers
  - EDA tools
- 1K–10K processor designs/year
- 5K–100K reference designs/year
- Millions of embedded products/year



The "Application-Specific" Drive

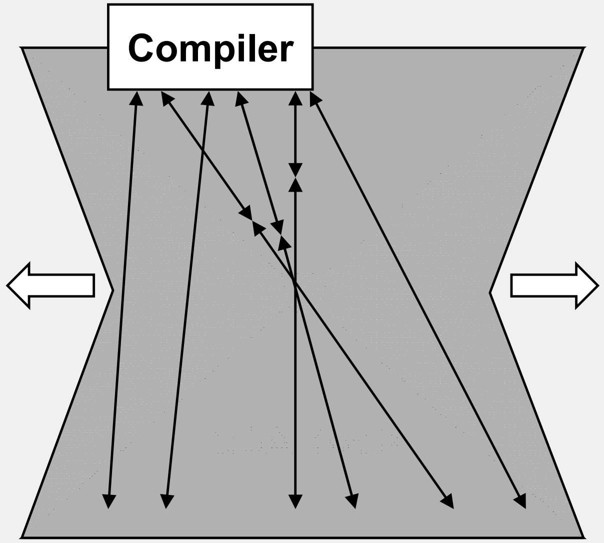
- **Application-specific processors, SoC's, MPSoC's**
  - **Special hardware functionality**
- **Application software development**
  - **Generic languages**
  - **C, C++, Java, Fortran-90…**



To enable *efficient* embedded-application development, EDA tools need to match architecture-specific functionality with the high-level programming environment

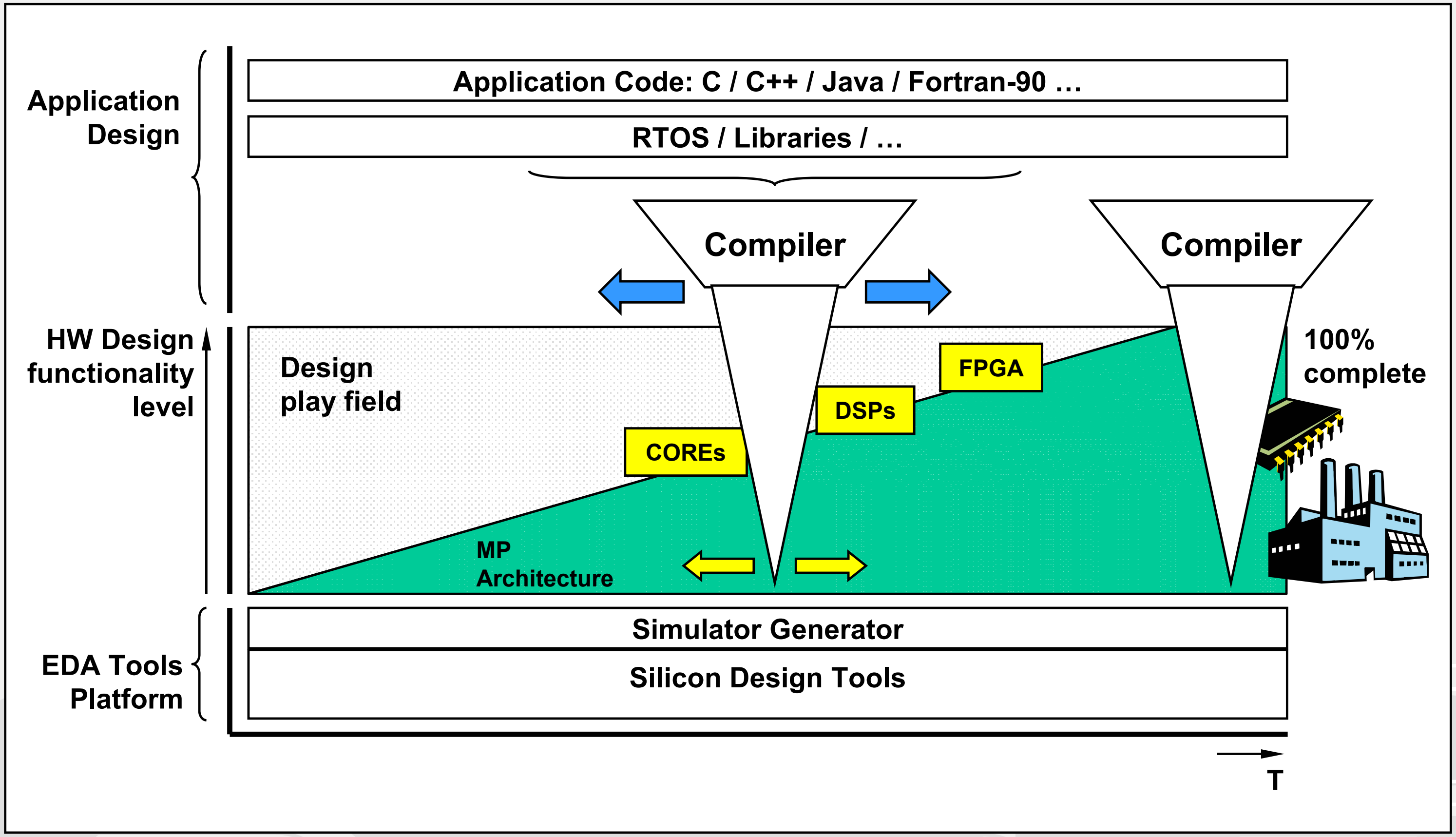
Compiler Technology is Crucial

- **Compiler technology is the natural bridge between high-level programming and hardware functionality**
- **Compiler should efficiently employ specific architectural features**



Flexible compiler technology is crucial to unleash MPSoC capabilities

Architecture Design Cycle



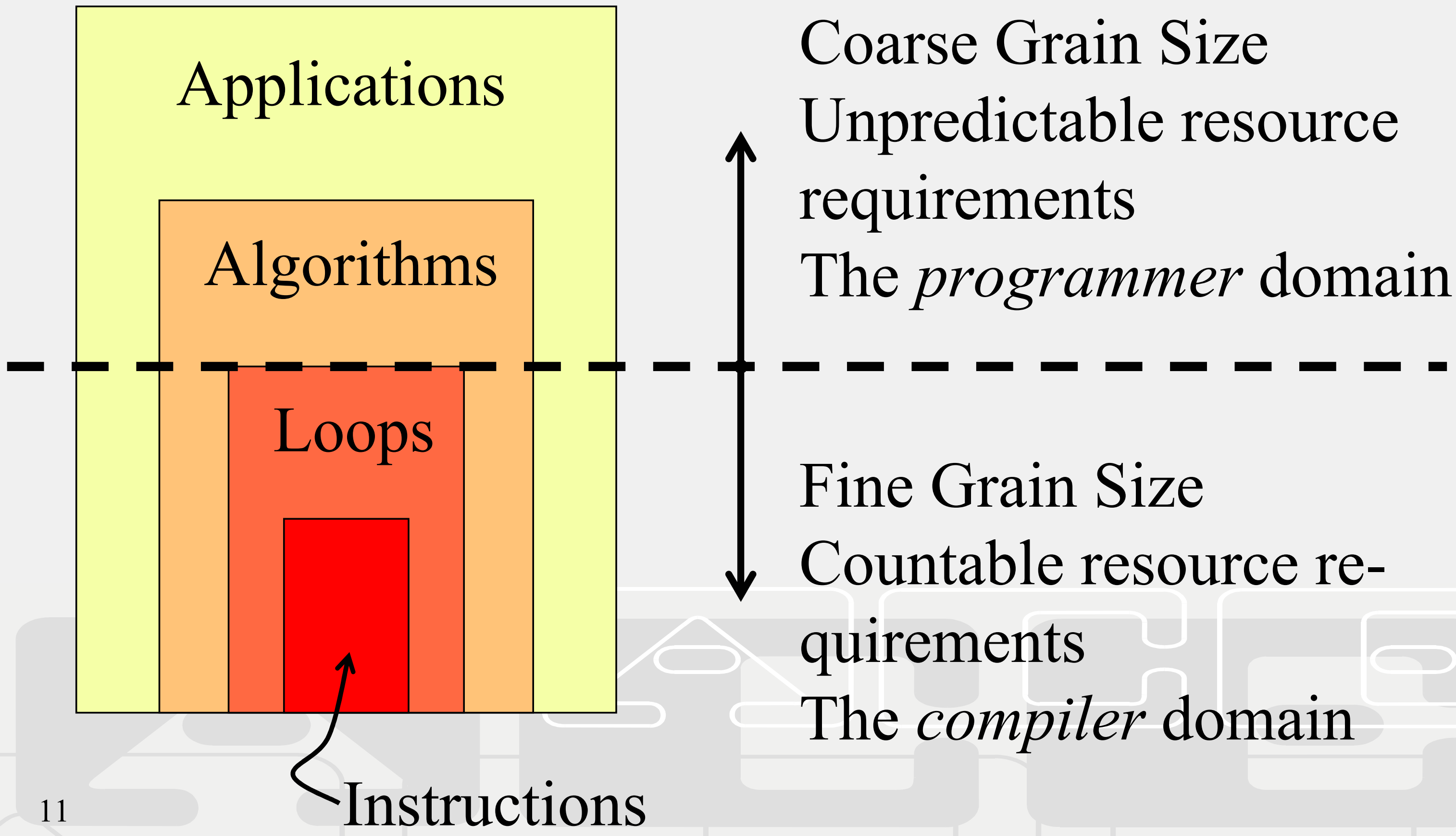
Parallel Architectures are Designed for Parallel Applications

- **The architecture's characteristics must match those of the application:**
  - **Grain size, the amount of work per task**
  - **Homogeneity of task size, computation, data distribution, communication**
- **The compiler, translating from application to architecture, must be able to represent these characteristics**

Grain Size and Homogeneity

- Differences can be orders of magnitude
- Fine grain size requires highly efficient task management
- Homogeneous tasks offer opportunity to share control and configuration overhead

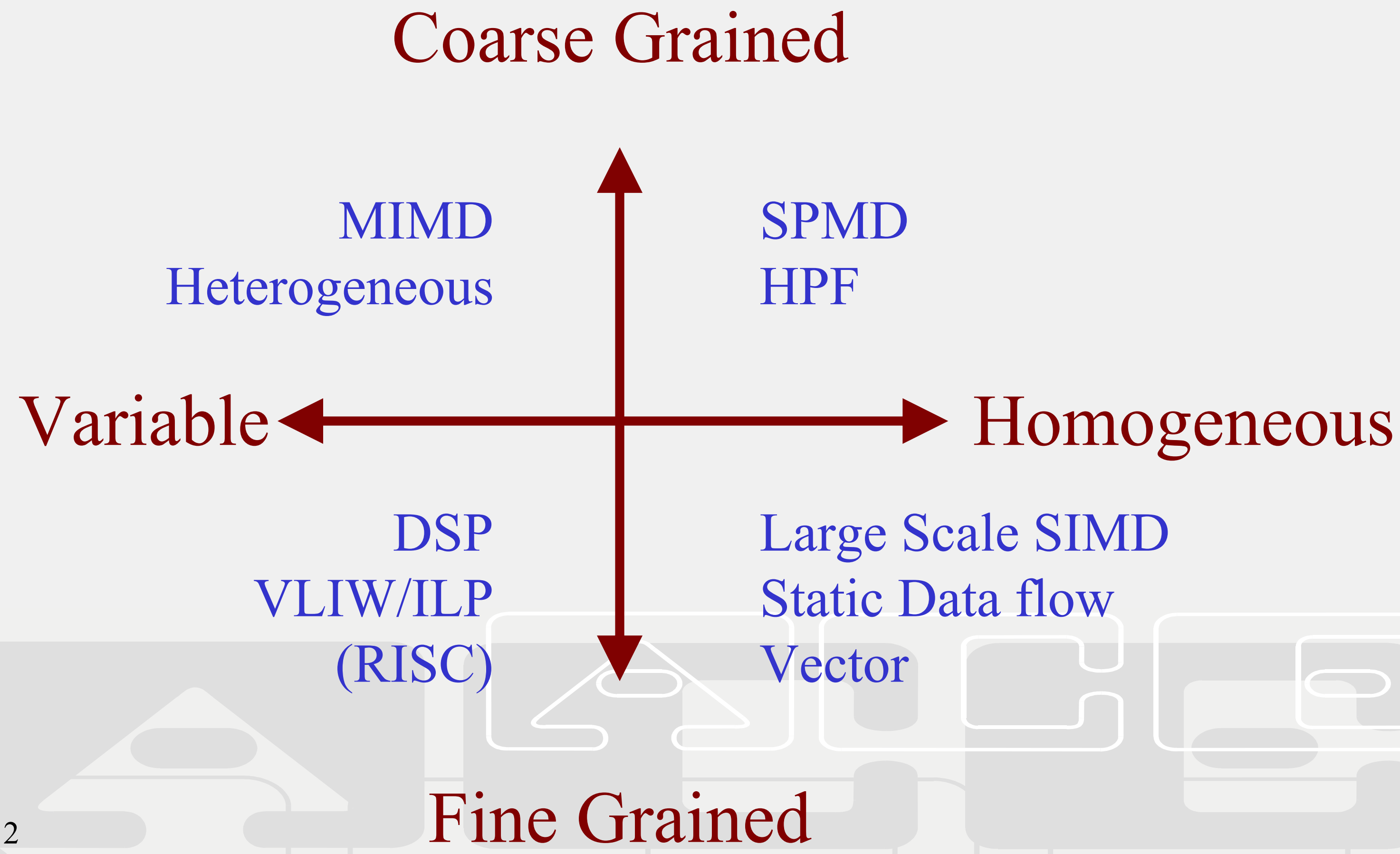
Smaller Grain Size is Easier to Handle by the Compiler



Coarse Grain Size
Unpredictable resource requirements
The *programmer* domain

Fine Grain Size
Countable resource re-quirements
The *compiler* domain

Architecture Characteristics

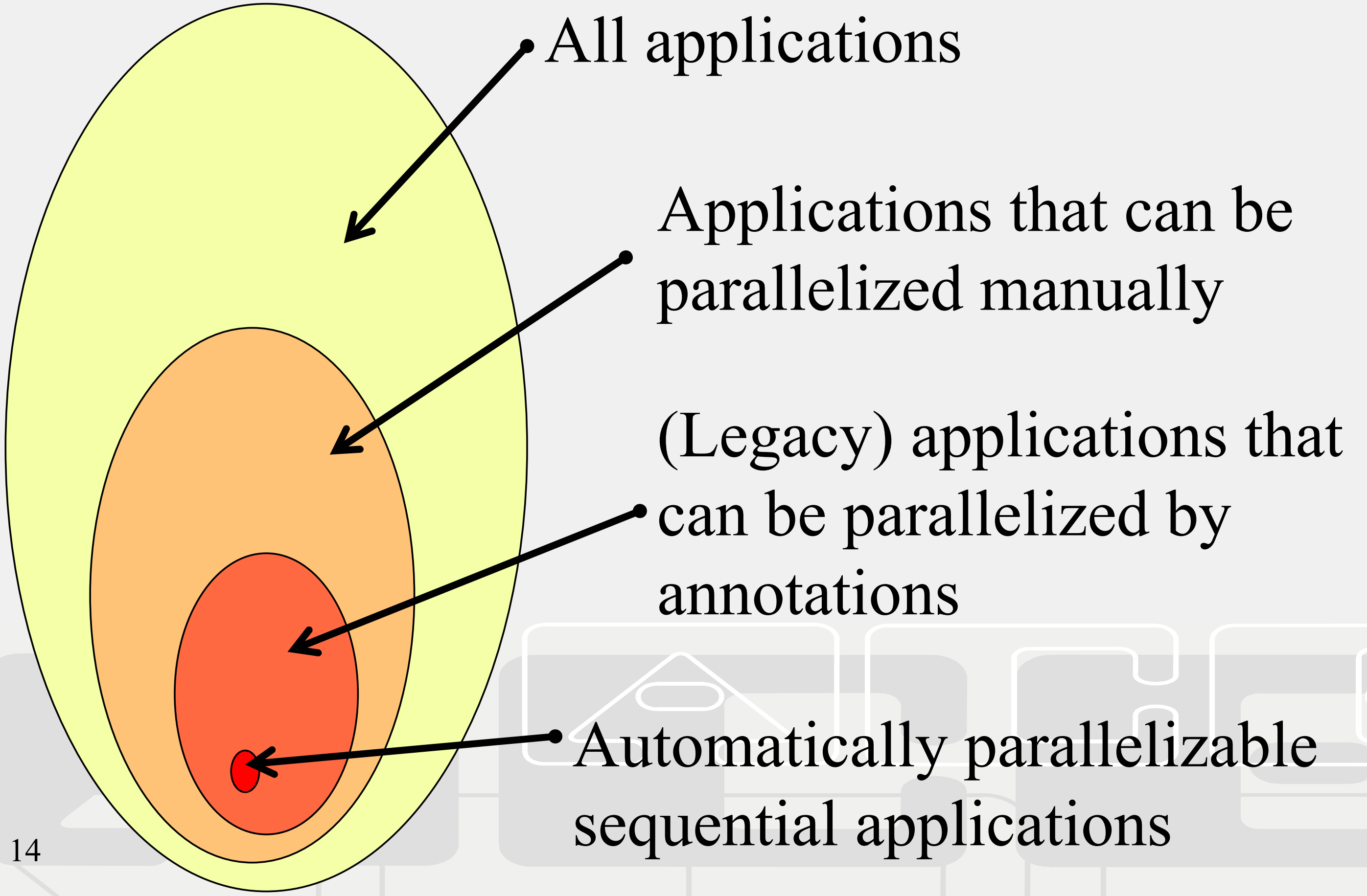


On Parallel Programming

- **Parallel programming is *extremely hard***
  - **Humans are not good at imagining all possible parallel execution orders**
  - **Parallel machines are non-deterministic, making testing and debugging very difficult**
  - **Explicitly parallel programs are architecture specific, hence hard to maintain and port**

- **The programming environment (compiler) should help as much as possible**
- **But…**

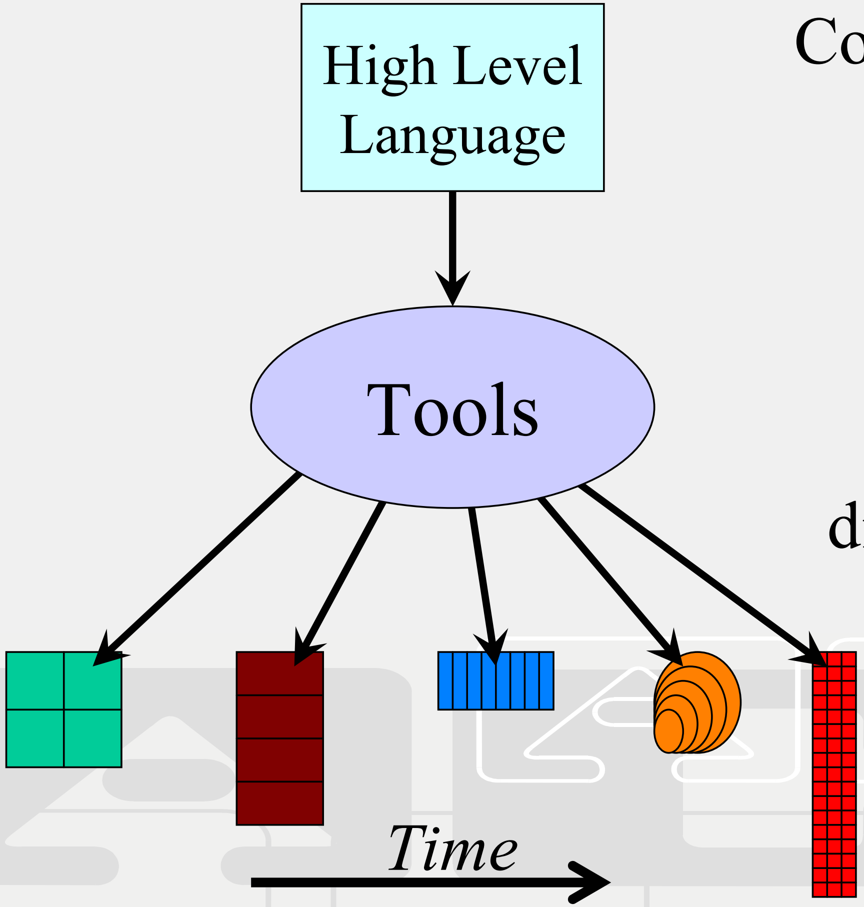
Applications & Parallelism


All applications
Applications that can be parallelized manually
(Legacy) applications that can be parallelized by annotations
Automatically parallelizable sequential applications

Directing Focus of Parallelization Efforts

- **Many applications cannot be efficiently parallelized at all**
  - **However, embedded (media) applications do offer lots of parallelism at several levels**
- **Automatically parallelizing sequential programs (holy grail) is interesting because it reduces programming effort, but only few programs qualify**

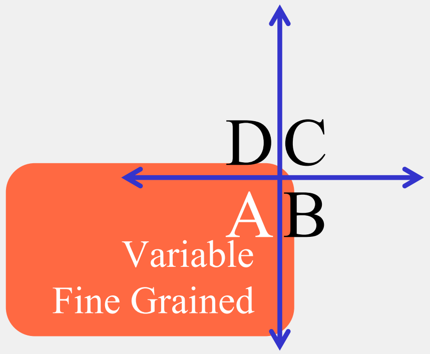
Programs and Languages Live Longer than Architectures



Compiler tools make the difference - *for parallel architectures even more so,* as they are more difficult to program

Architectures

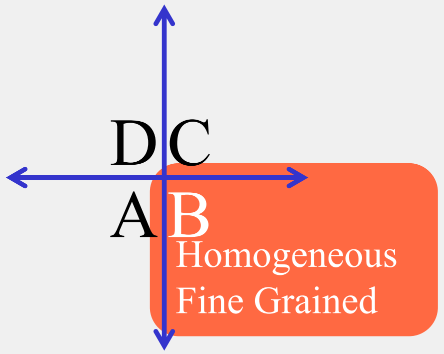
Type *A* Parallelism: Variable, Fine Grained



- Sequential programs, with annotations (restrict, memory spaces) to help the compiler
- Automatically parallelized by the compiler for pipelined, DSP and VLIW/ILP architectures

Type *B*: Homogeneous, Fine Grained

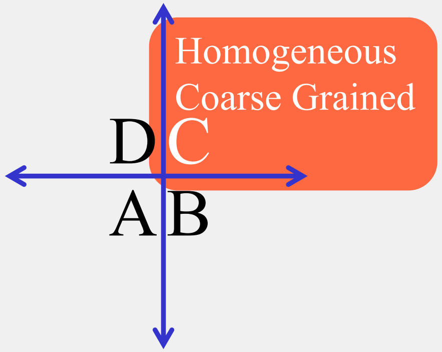
- Needs *vectorizing* compilers
- Resources still predictable
- Vectorizing plain C, with pointers, is hard; annotations help



Architectures	Compilers
-(Large) Vector	-Allocation of resources
-SIMD (small vector)	-Tiling of arbitrary data-size
-Fine grain static data flow (aka reconfigurable)	-Streams paradigm

Type *C*: Homogeneous, Coarse Grained

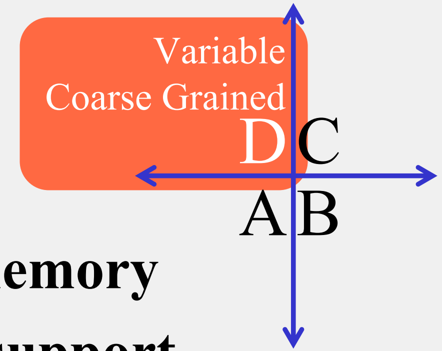
- **NPU architectures: static control and communication patterns**
- **Replicated processor architectures**



Programming Paradigms	Compilers
-Explicit: OpenMP, StreamIt -Performance analysis based resource allocation -SPMD, HPF (not streamed)	-Generate and optimize global communication and control -Abstract interpretation

Type *D*: Variable, Coarse Grained

- MPSoC architectures, collection of specialist processors
- Differentiate between networked (uncommon in embedded) and shared memory
- Programming paradigms and compiler support similar (add MPI) to Type C, but more complex because of heterogeneous architecture



Evolutionary: MPSoC Parallelism Type: *D*

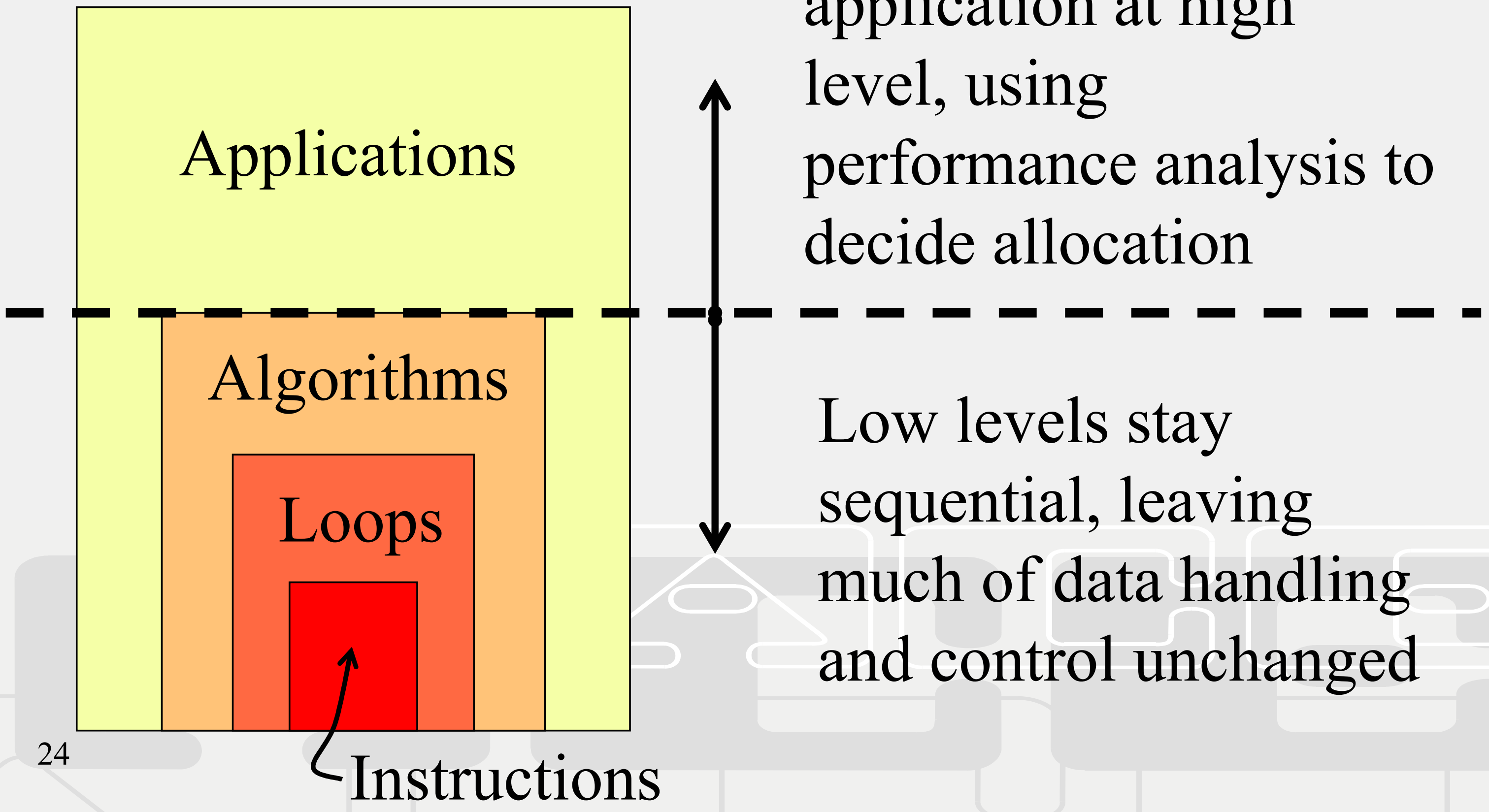
System on a Chip Evolutionary Approach

- Chip space is readily available, but non-locality of on-chip communication is a bottleneck
- Two processors of 300MHz are more power efficient than one of 600MHz ($\text{Freq}^3 \approx \text{Power}$)
- Hardware/System design tools “easily” support putting legacy, specialized, building blocks together
- But programming SoCs proves *cumbersome*

SoC Programming Problems

- Existing, single processor, programs are not written to serve as a parallel component
- Application data-flow and control-flow are separated
- Distributed control of heterogeneous independent processors requires unified run time support
- Multiple data-models (24 versus 32 bit integers)
- As applications become more diverse, patterns of parallelism become more dynamic

Parallelization of Programs for SoC



Programmer splits application at high level, using performance analysis to decide allocation

Low levels stay sequential, leaving much of data handling and control unchanged

Supporting Parallelization for SoC

- Programmer decides based on *performance analysis* feedback, then uses *annotations* to distribute the application across multiple processors
- Compiler can generate data sharing and control “glue”, allowing programmer more flexibility in decisions
- Compiler can compile for heterogeneous system, taking care of differences in data models
- Method is suitable for legacy programs that can be split at high level

Alternative SoC Parallel Programming Paradigm

- Explicitly parallel MPI-like model where data transfer and program control is based on message passing
- Compiler can optimize message passing to exploit shared memory (avoiding copies)
- Application requires parametrization to allow experimentation and to adjust to optimal mapping to parallel hardware --- *abstract interpretation* can hard-wire parameters at compile time to avoid run-time overhead
- But this will push programmers into a certain framework --- programmers may resist against this

Where are we now

- Assembly is dead long live assembly
- C/C++ codes are very expensive to reengineer
- Compilers cannot identify sufficient parallelism
- Design money is drained into H/W design
- Programmers don’t like non-orthogonal paradigms
- Codes exist and shall be reused

Complexity?

- **Flexible x reconfigurable x multi-CPU x NUMA x SHM x SIMD x MIMD x SPMD x distributed memory x heterogeneity x data models x multi-tasking x message passing x CSP x data parallelism x monitors x ILP x VLIW =**

NOT WORKING

Conclusions

- KISS
- Let more design money flow to software
- Allow for 10 years of research
- Agree on methods and paradigms (not standards)
- Stop h/w designers from being ‘clever’
- History is there for a reason
- The dinosaurs will not die suddenly