

Programmability of multi-core architectures

Jos van Eijndhoven

CTO & Co-founder

MPSoC 2014

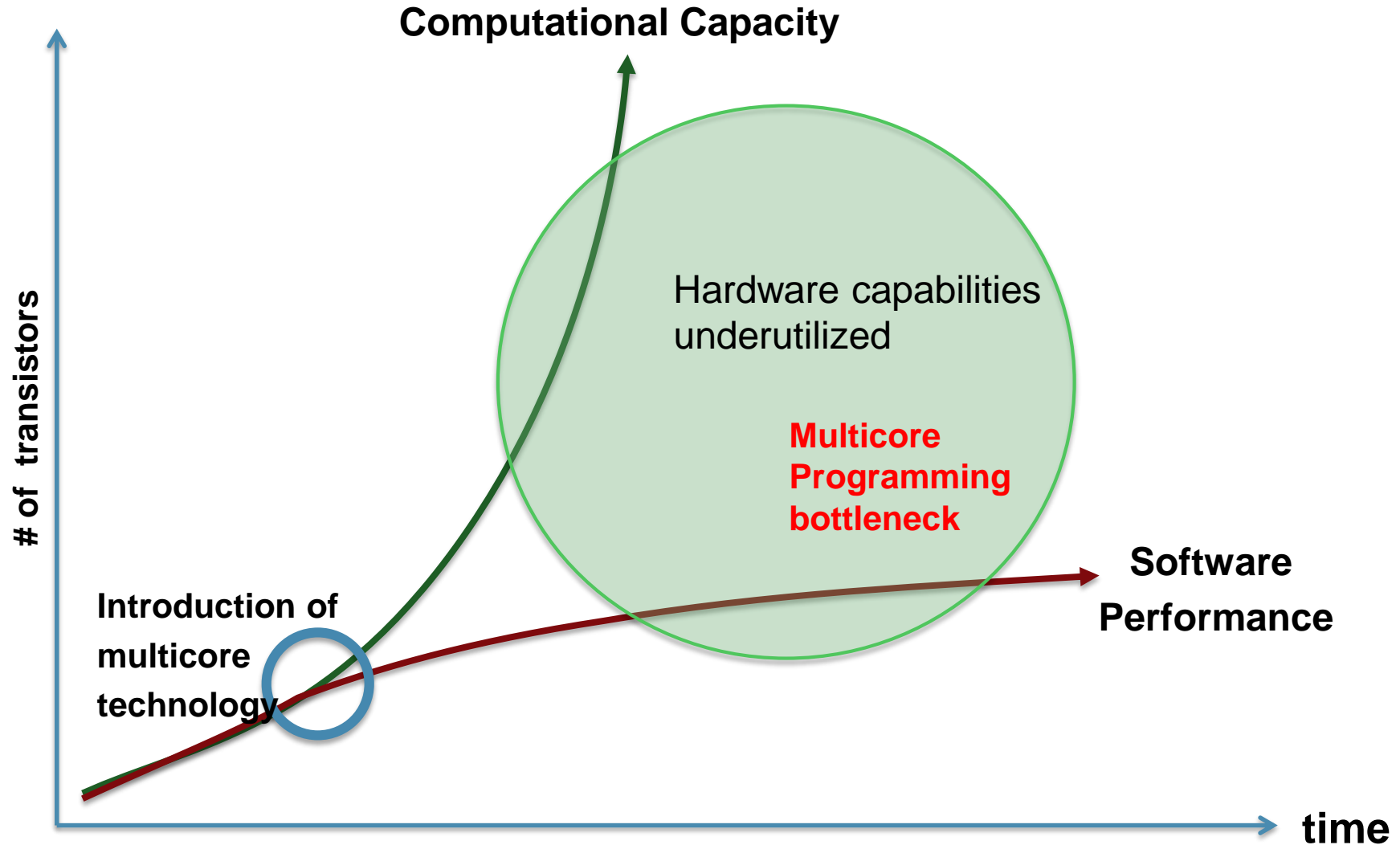
Margaux, France, July 2014



jos@vectorfabrics.com

Eindhoven, The Netherlands

Moore's law versus Amdahl's law



What about parallel programming bugs?

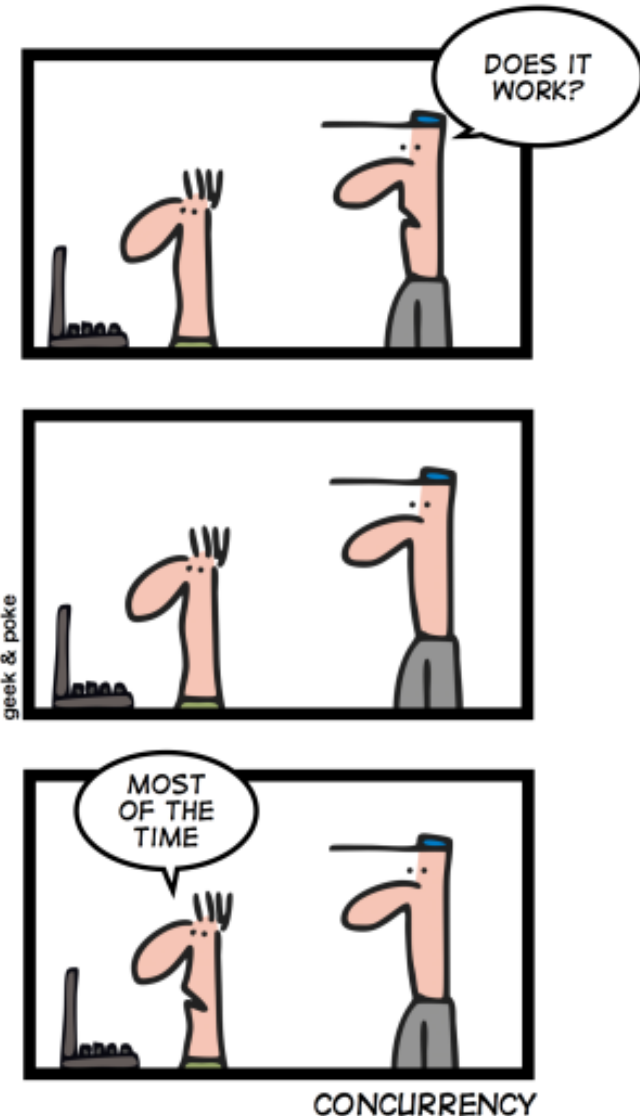
Global `int x = 0, y = 0;`

Launch four threads, namely:

- Thread 1: `x = 1;`
- Thread 2: `y = 1;`
- Thread 3:
`if (x && !y) print("X first");`
- Thread 4:
`if (y && !x) print("Y first");`

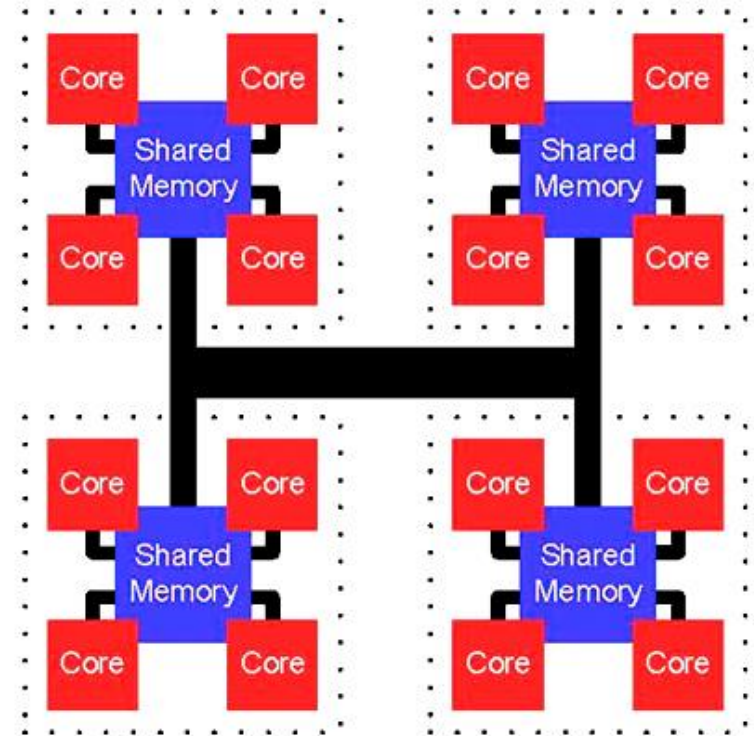
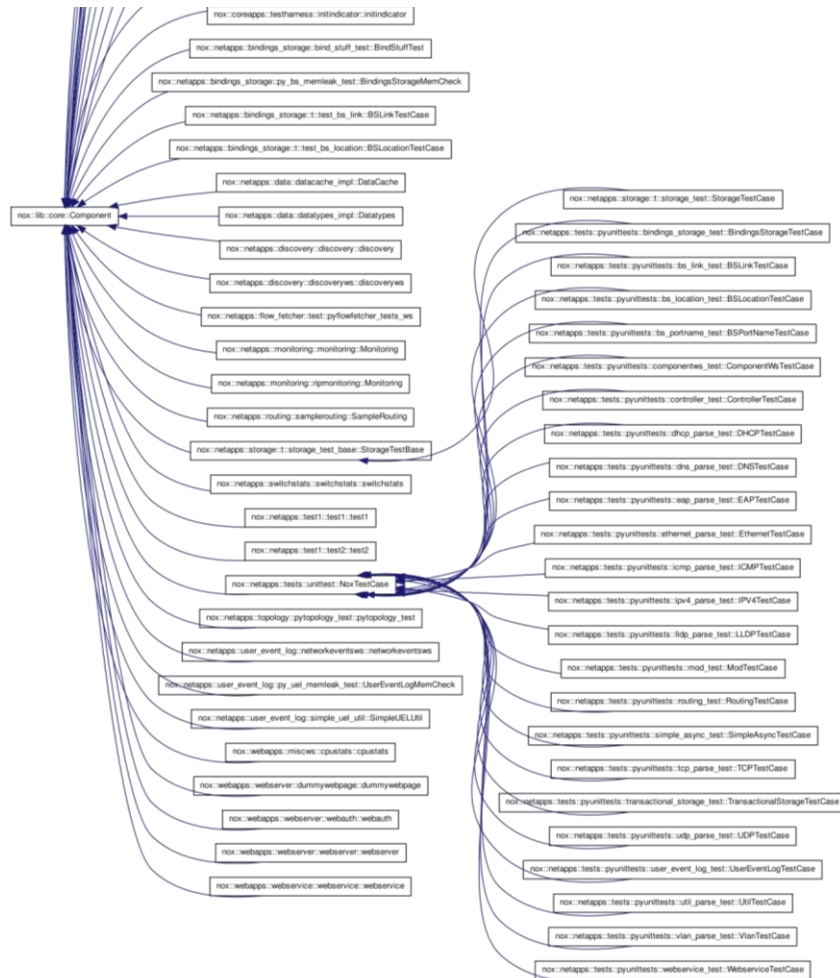
Memory ordering issues result in
nondeterministic behavior!

Leads to hard-to-find bugs!



Partitioned memory?

Embedded products: growing complexity of code and data



Strong foundation – learn from history?

- Shared-memory multi-cores
- Cache-coherent
- Virtual memory paging
- Single shared OS kernel

Successful abstraction!

Proven by 30 years of collaborative
compute architecture history

IBM, SUN, HP, Intel, AMD, ARM ...

Be careful to create proprietary
non-standard solutions!



Classic C/C++ for multi-threading

Three basic primitives, and some OS-level functionality

- Volatile variable declarations:
force compiler load/store generation, limit compiler re-orderings
- Memory fence operations:
force load/store ordering at runtime in the memory system
- Atomic operations:
indivisible read-modify-write (increment, test-and-set)
- Higher-level constructs (semaphores, condition variables) that include OS and kernel support → thread sleep and wakeup

Only 'volatile' is standardized in C/C++. Originally designed for I/O to hardware.

Posix thread library in 1995, fences/atomics are compiler specific intrinsics

Classic C/C++ constructs for threading

Three basic primitives, and some OS-level functionality

- Volatile variable declarations:
force compiler to store generation of compiler re-orderings
- Memory fence operations:
force load/store ordering in the memory system
- Atomic operations:
indivisible read-modify-write (increment, test-and-set)
- Higher-level abstractions (semaphores, condition variables) that include OS and kernel support → thread sleep and wakeup

Get rid of all of this 15 years of programming practice
bold move by the C++11 committee!

C11/C++11 parallel programming

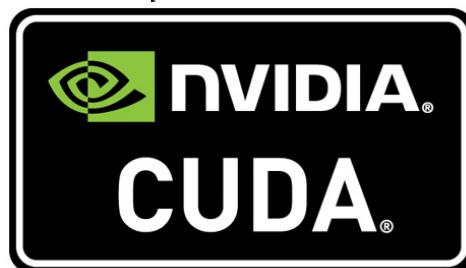
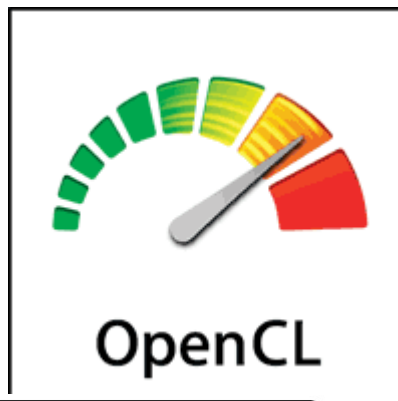
Creation of multi-threaded programs:

- The C/C++ compiler will **always** assume multi-threaded access to variables with global scope. This inhibits some optimizations. (C++11 has no 'volatile' to denote inter-thread data exchange)
- **Atomic** operations are overloaded with **memory fence** behaviors. These are the basic building blocks for inter-thread synchronization.

Finally: multi-core memory behavior is specified for C/C++ !
it standardizes on a weak memory ordering!

Your MPSoC adheres to this memory ordering semantics?

Too many paradigms...



Semiconductor vendor's sales challenge



Multicore vendor

Look at the raw performance and power features of our **multicore chip**!

What is the cost of **porting my code** to your multicore?

Show me what the **performance and power** of **my code** will be on your multicore

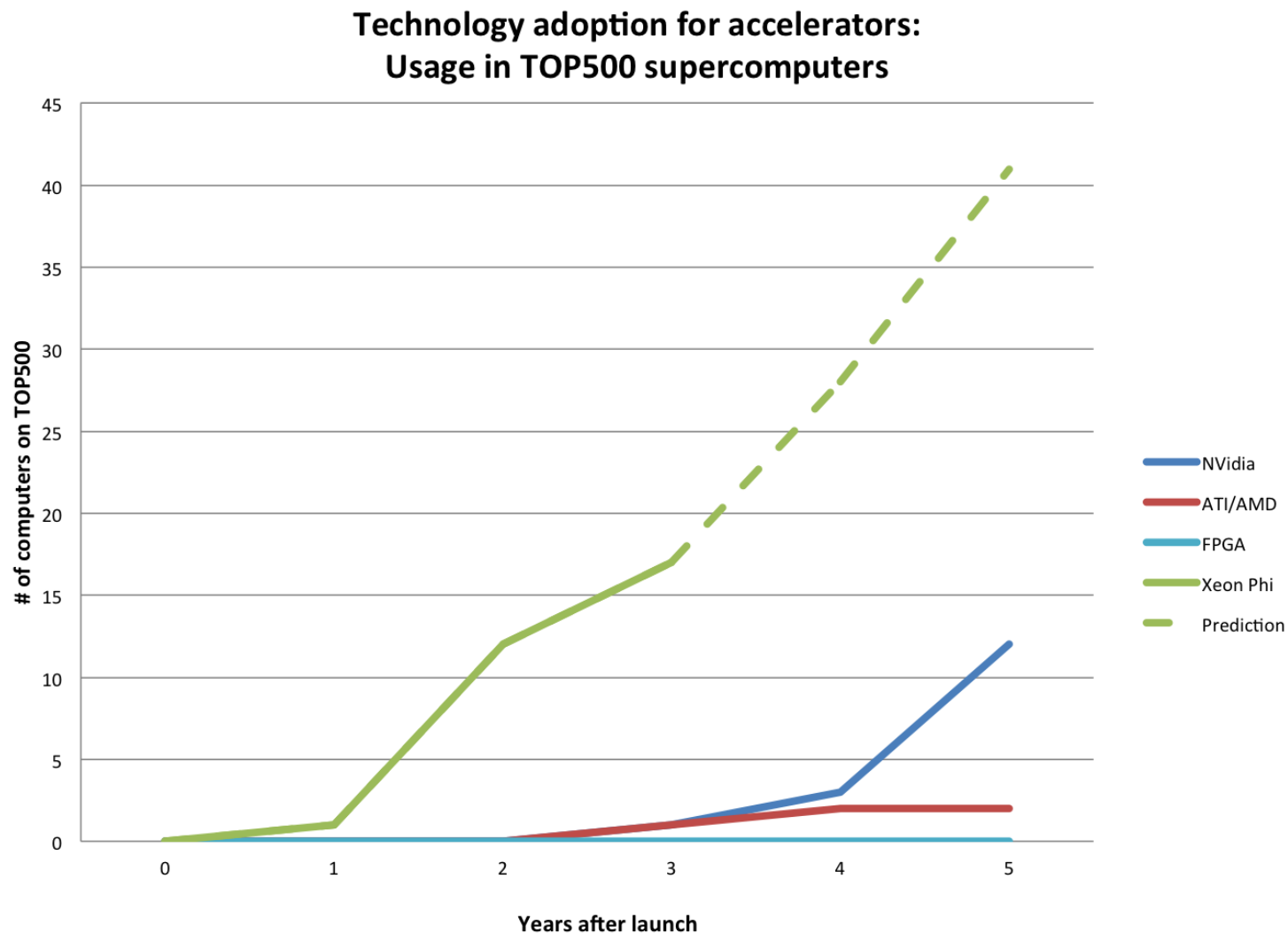
I need **support**! I don't understand how to benefit from the hardware features through my code



Device OEM

Good programmability is required to win enough customers!

Value of programmability: fast adoption



PAREON: performance analysis

The screenshot displays the PAREON user interface, which is divided into several panels for performance analysis.

Profile Panel (Left): A table showing the execution profile of various components. The components are listed under the 'Name' column, and their 'Coverage' and 'Delay' are shown in the adjacent columns. The 'Loop_32' component is highlighted in blue.

Name	Coverage	Delay
sgetf2_	72	71.93
xerbla_		0.00
slamch_	100	0.00
Loop_32	81	71.92
sgetrs_	57	3.67
printf		0.10
Loop_238	100	0.02
printf		0.05
printf		0.03

Properties Panel (Bottom Left): A table showing the properties of the selected component, 'Loop_32 (sgetf2_)'. The properties are listed under the 'Property' column, and their values are shown in the adjacent column.

Property	Value
Loop	Loop_32 (sgetf2_)
#invocations	1
#iterations / invocation	100 (iteration histogram)
Invocation time	3.9 ms (39.0 us / iteration) (time hist)
Memory penalty	94.2 us (2.4 %)
Data cache penalties	92.5 us (2.4 %)
Data cache misses	
Level 1	11733 (1.1 %)
Level 2	8 (0.0 %)
DRAM traffic	512 B (128 KiB/s)
Load count	717340
Store count	352894
Mapped to Instance	Cpu0[0]
Source location	sgetf2.c:141-185
Line coverage	80.8 %
Uncovered lines	

2D-Profile Panel (Center): A 3D bar chart showing the execution profile of the selected component, 'Loop_32'. The chart displays the execution time of different parts of the loop, with a callout indicating that loop-carried dependencies hinder parallel execution of loop iterations.

Dynamic Help Panel (Right): A panel providing information about the 2D-Profile view, including a guide on how to use the view and a list of dependencies.

Callouts:

- View on call tree with relative workload**: A callout pointing to the 2D-Profile view.
- Loop-carried dependencies hinder parallel execution of loop iterations**: A callout pointing to the 2D-Profile view.
- Other performance statistics: Iteration counts, cache penalties**: A callout pointing to the Properties panel.

PAREON: Schedule data dependencies

The screenshot displays the PAREON software interface, which is used for analyzing and scheduling data dependencies. The interface is divided into several panels:

- Profile / Partitions:** Shows partitioning candidates for Loop_38. The CPU data partitioning - vfTasks section indicates 4 threads, with a global speedup of 2.3 and a global overhead of 6%. A table lists the partitioning results for Loop_38, showing a speedup of 3.9 and an overhead of 1%.
- Properties / My changes:** Displays properties for Loop_38 (sgtf2_). The iteration count is 150. Iteration statistics show a computation time of 85.3 us (92.7%), a memory penalty of 6.8 us (7.3%), a load count of 15770, a store count of 7802, and an instruction count of 104658. The source location is sgtf2.c:141-185.
- 2D-Profile / Schedule:** Shows a 2D profile of data dependencies. The top bar indicates "Execution - 99 %". The main area displays a schedule execution (prologue - steady state - epilogue) with a grid of iterations. A blue box highlights a specific iteration, and a callout points to it with the text "Estimate multi-thread fork/join overhead".

Obtain a **preview** on a potential parallelization assume synchronization on complex dependencies

Pareon technology

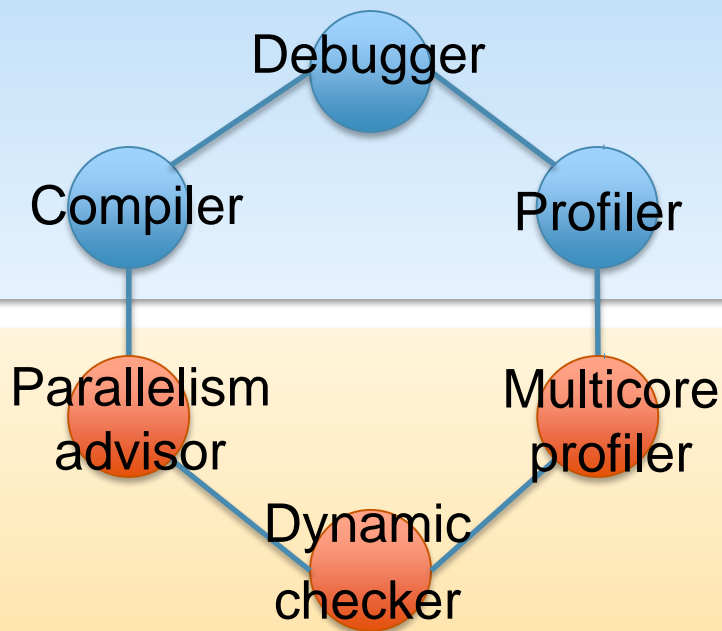
- Compile C/C++ sources with instrumenting compiler
- Hijack calls to OS and standard libraries through runtime binary rewrite
- Application execution creates event trace
- Analyze event trace for effect on (some other) target platform

Works for real-world code:

- Google Chromium browser: 10M+ lines of C++, parallelized loop that contains 1M+ lines of code.
- Game engine with 500Klines of code

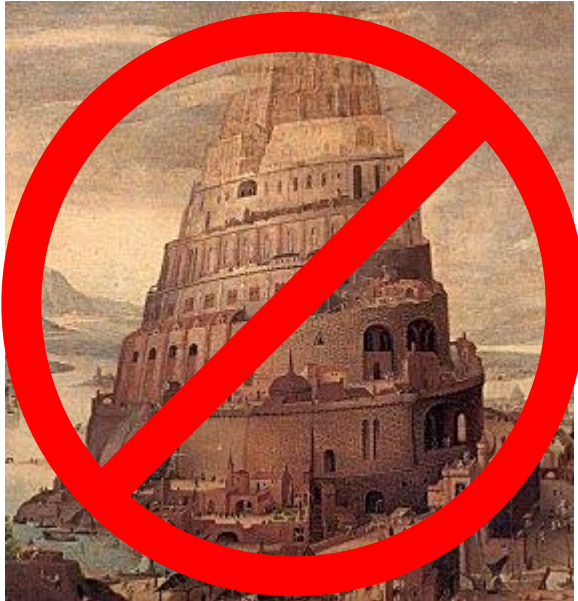
Software development kit – more tools?

Current software development kit



- Basic SDK focuses on programming on just one core
 - Multicore tools needed for interactions between cores
- **Parallelism advisor** – insight in how to partition and map code & data structures
- **Multicore profiler** – optimize scheduling and memory accesses
- **Dynamic checker** – check for multicore errors such as data races, deadlocks

Takeaway



1. Multicore programming challenge hampers adoption of platforms
2. There is no unified manycore programming paradigm, only partial solutions
3. Programmers need further multicore programming tools in their SDK

Create multicore chips for programmers,
not for hardware designers

Products Services News Company

Performance

Potential Performance

Realized Performance

Adding more hardware

VectorFabrics

Unleash the potential of your product's performance.

We'll help you get where you need to be.

[See how](#)

VectorFabrics

PRODUCTS

Pareon makes your C/C++ code run faster

CONSULTANCY

Software optimization

TRAINING

Multicore programming training

PARION OVERVIEW >

CHECK OUT OUR EXPERTISE >

SEE OUR TRAINING PROGRAM >

Optimize your software

<http://vectorfabrics.com>



Jos van Eijndhoven
jos@vectorfabrics.com

+31 40 8200960

Eindhoven, The Netherlands