Accelerating Scientific Computing with Massively Parallel Computer Architectures

IMPRS Winter School, Wroclaw

Jun.-Prof. Dr. Christian Plessl

Custom Computing
University of Paderborn

Outline

- history and trends in massively parallel computer architectures
- comparative case studies
- simplifying the use of massively parallel architectures
- conclusions and outlook

About me

- mini CV
  - 2001 Diploma in Electrical Engineering, ETH Zürich
  - 2006 PhD in Information technology and electrical engineering, ETH Zürich
  - 2006-2007 Postdoc, ETH Zürich
  - 2007-2011 Postdoc and research group leader, University of Paderborn
  - since 9/2011 Assistant professor for Custom Computing, University Paderborn + head of custom computing at Paderborn Center for Parallel Computing

- research interests
  - computer architecture
  - custom computing (architecture, design methods and tools, runtime systems)
  - adaptive and self-optimizing computing systems
  - application of massively parallel computing in science and high-performance embedded systems

Outline

- **history and trends in massively parallel computer architectures**
- comparative case studies
- simplifying the use of massively parallel architectures
- conclusions and outlook

Motivation: Increasing Demand for Computing

- computer simulation has been established as a standard method in many areas of science and engineering
  - e.g. computational fluid dynamics, structure simulation, propagation of electromagnetic fields, ...
  - demand for finer temporal and spatial resolution or more complex models
  - in many cases compute bound

- new areas are addressed with different characteristics
  - e.g. bioinformatics, large scale molecular dynamics, system's biology, ...
  - compute but increasingly also memory bound (big data)

- availability of high performance computing resources is a competitive advantage or necessity in many areas

Simple CPU Performance Model (1)

- performance measured as execution time for given program

$$t_{execution} = \#instructions \cdot \frac{cycles}{instruction} \cdot \frac{time}{cycle}$$

reduce #instructions by increasing work per instruction

$a \leftarrow a + b \times c$

mac $a $b $c

multiply-accumulate instruction

c[3..0] = a[3..0] + b[3..0]

a	a3	a2	a1	a0
	+	+	+	+
b	b3	b2	b1	b0
	=	=	=	=
c	a3+b3	a2+b2	a1+b1	a0+b0

SIMD/vector instructions

Simple CPU Performance Model (2)

- performance measured as execution time for given program

$$t_{execution} = \#instructions \cdot \frac{cycles}{instruction} \cdot \frac{time}{cycle}$$

reduce #cycles/instructions (improve throughput)

```
a = b + c
d = a + e
```

overlap execution of instructions (pipelining)

begin execution of dependent instructions already before all dependencies are resolved

```
a = b + c
d = e + f
```

start multiple instructions at once (multiple issue/superscalar)

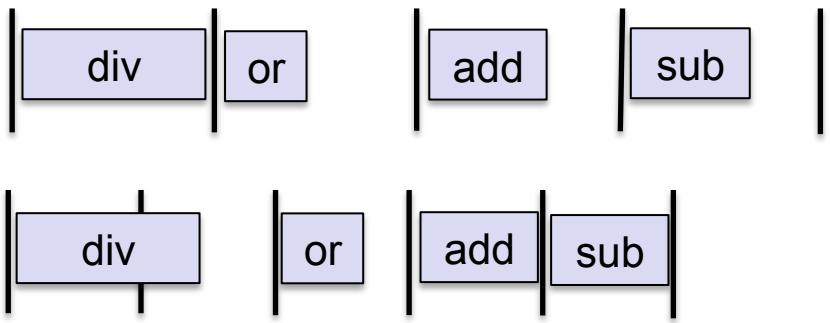
concurrently start independent instructions

Simple CPU Performance Model (3)

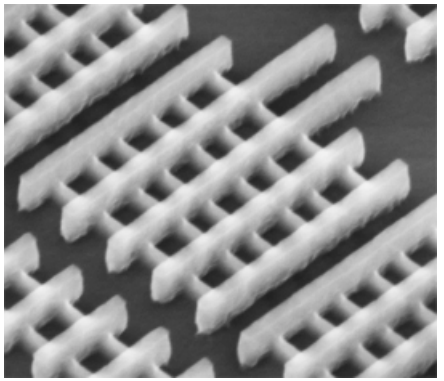
- performance measured as execution time for given program

$$t_{execution} = \#instructions \cdot \frac{cycles}{instruction} \cdot \boxed{\frac{time}{cycle}}$$

execute cycles in shorter time



balance time spent in each pipeline stage



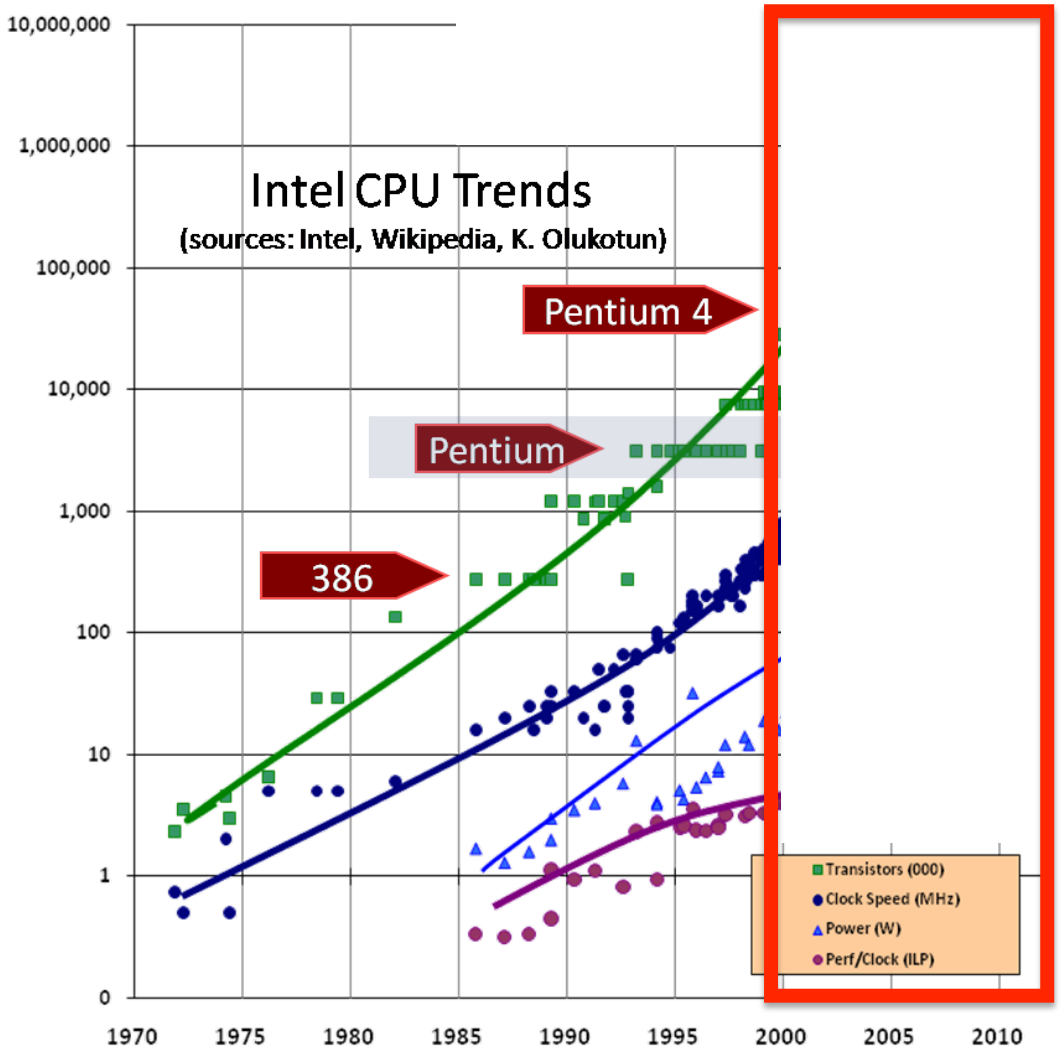
use leading semiconductor technology

image: Intel

Riding the Waves of Moore's Law

- despite all computer architecture innovations, Moore's law contributed most to performance increase
  - more usable chip area
  - faster clock speed
  - exponential growth of performance

- but since early 2000's hardly any increase in single core performance due to
  - power dissipation
  - design complexity



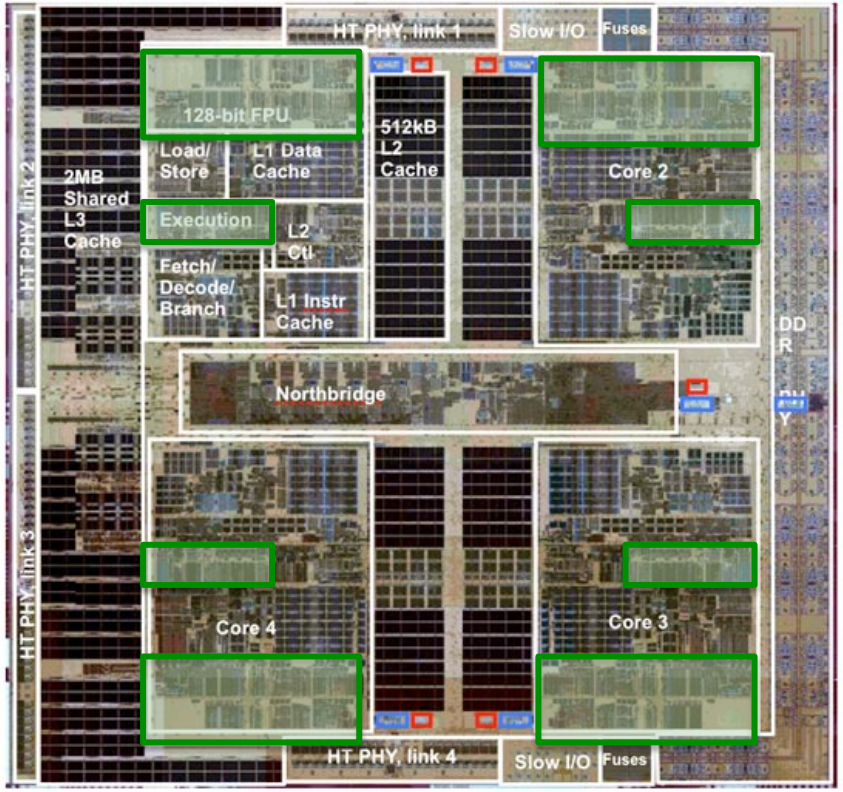
source: Herb Sutter, Microsoft

Limits of Single-Core CPU Performance Growth

- CPUs not tailored to particular task
  - unknown applications
  - unknown program flow
  - unknown memory access patterns
  - unknown parallelism

- generality causes inefficiencies
  - poor relation of active chip area to total area
  - excessive power consumption
  - difficult design process

die shot of a 4-core AMD Barcelona CPU



chip area that contributes to actual computation

image: Anandtech

Parallel Computing Comes to Rescue

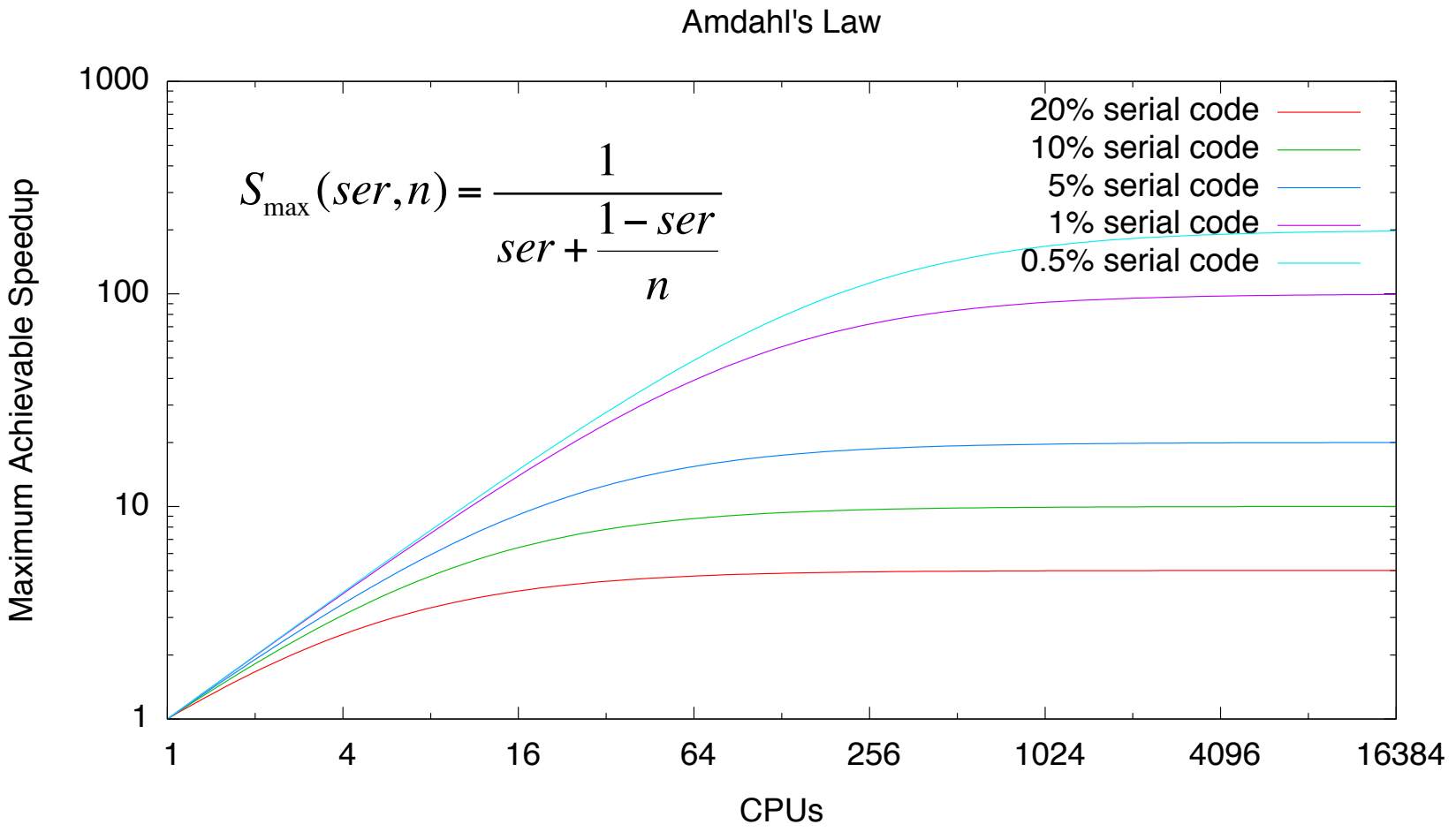
- parallel computing is celebrating a renaissance
  - single core performance is stagnant
  - today processor cores can be integrated on a single chip
  - fast networking is available (clusters)

- promise of parallel computing
  - divide and spread work to N processors to achieve an N-times speedup
  - ideally: use N processors to achieve a speedup of factor N
  - improved energy efficiency since processors can be used that provide the best performance / energy ratio

- practical challenges
  - how to program parallel processors (model, language, compiler, ...)
  - management of data and I/O
  - overheads of parallelization

Amdahl's Law – Limits to Parallel Computing

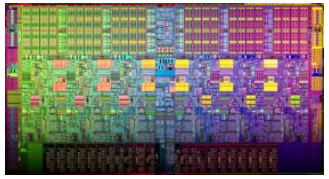
- fraction of non-parallelizable (serial) part of program limits maximum acceleration when using N processors



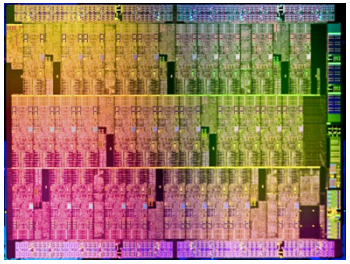
Parallel Processor Architectures

- many different processor architectures have been recently introduced
  - typically used as hardware accelerators in combination with regular CPUs
- interesting times for computer architecture research
  - a lot of experimentation: many simple vs. few complex cores, shared vs. distributed memory, fixed function vs. customizable, ..
  - the future may be heterogeneous
- current major trends
  - general-purpose graphics processing units (GPGPUs)
  - custom computing / field-programmable gate arrays (FPGAs)
  - many-cores e.g. Intel SCC, Intel MIC, Tilera, ...

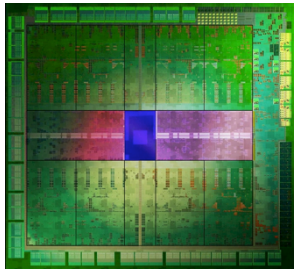
Current Parallel Computing Architectures



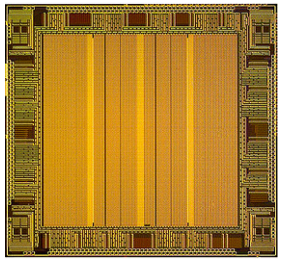
multi-core CPU



many-core CPU



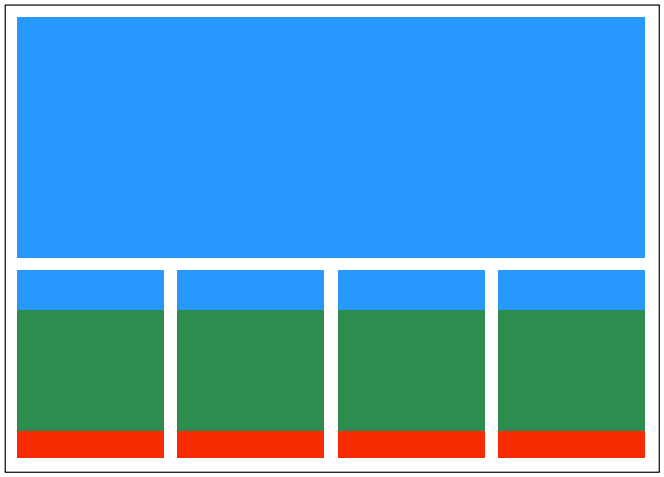
graphics processing units (GPU)



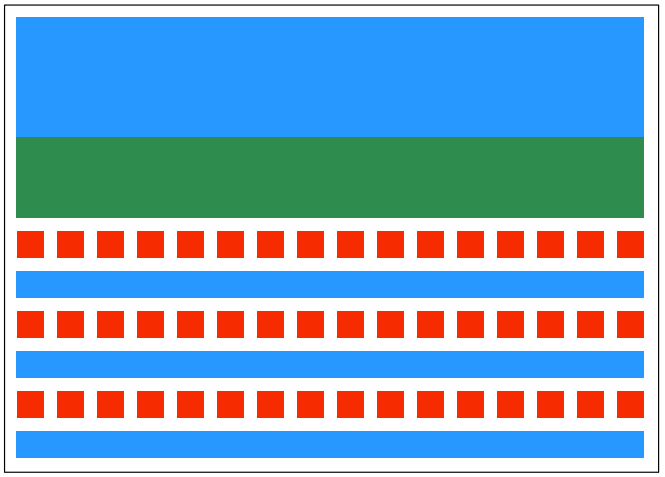
field programmable gate arrays (FPGA)

cores	~10	~100	~1000	~100'000
core complexity	complex (opt. for single-thread performance)			simple
computation model	MIMD + SIMD		SIMD	data-flow
parallelism	thread and data parallel		data parallel	arbitrary
memory model	shared	distributed		
power	150W	200W	250W	50W

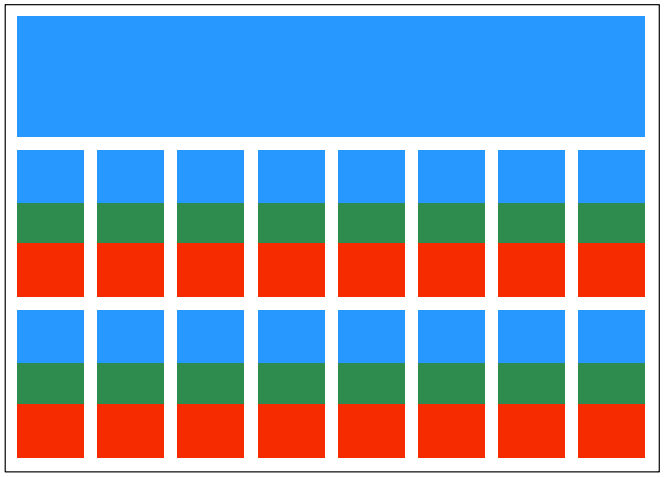
Schematic View of Massively Parallel Architectures



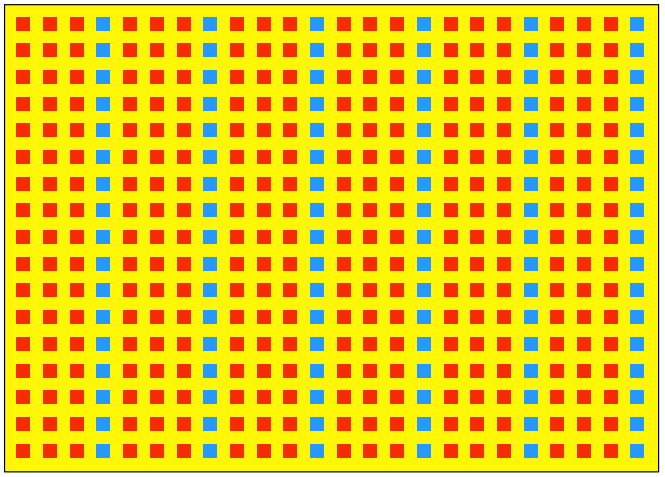
multi-core CPU



GPU



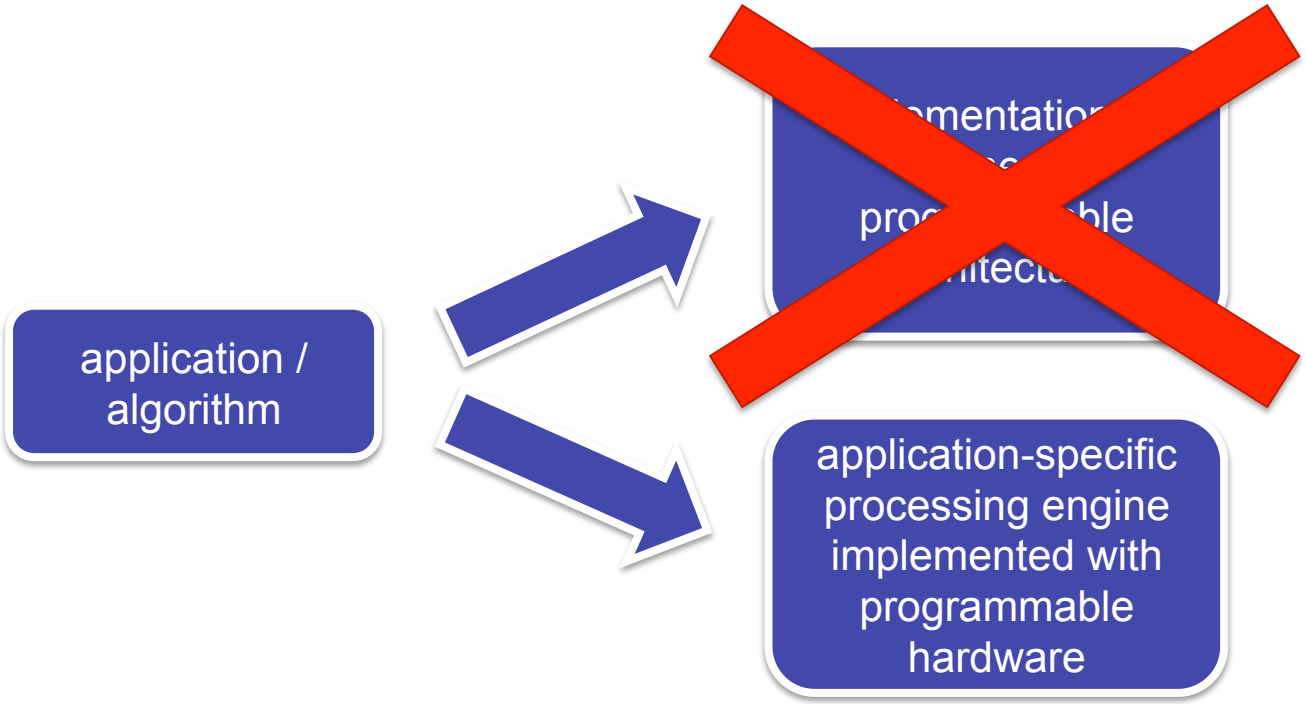
many-core CPU



FPGA

computational unit execution controller interconnection network on-chip memory

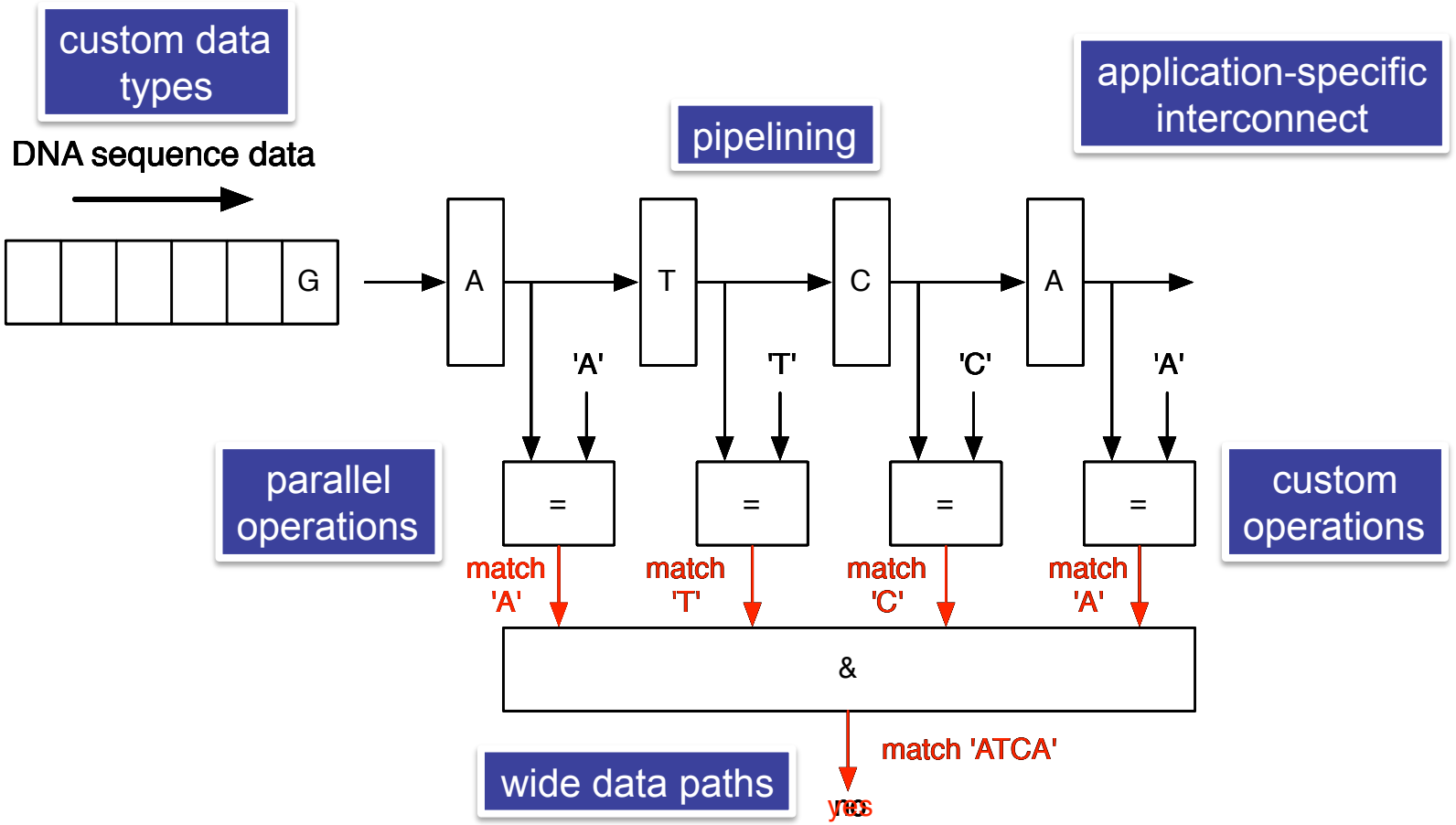
Custom Computing with FPGAs – Basic Idea



- challenges
  - translation of application into a custom hardware accelerator
  - development of efficient programmable hardware for implementing accelerators
  - enabling wide range of users to perform this task

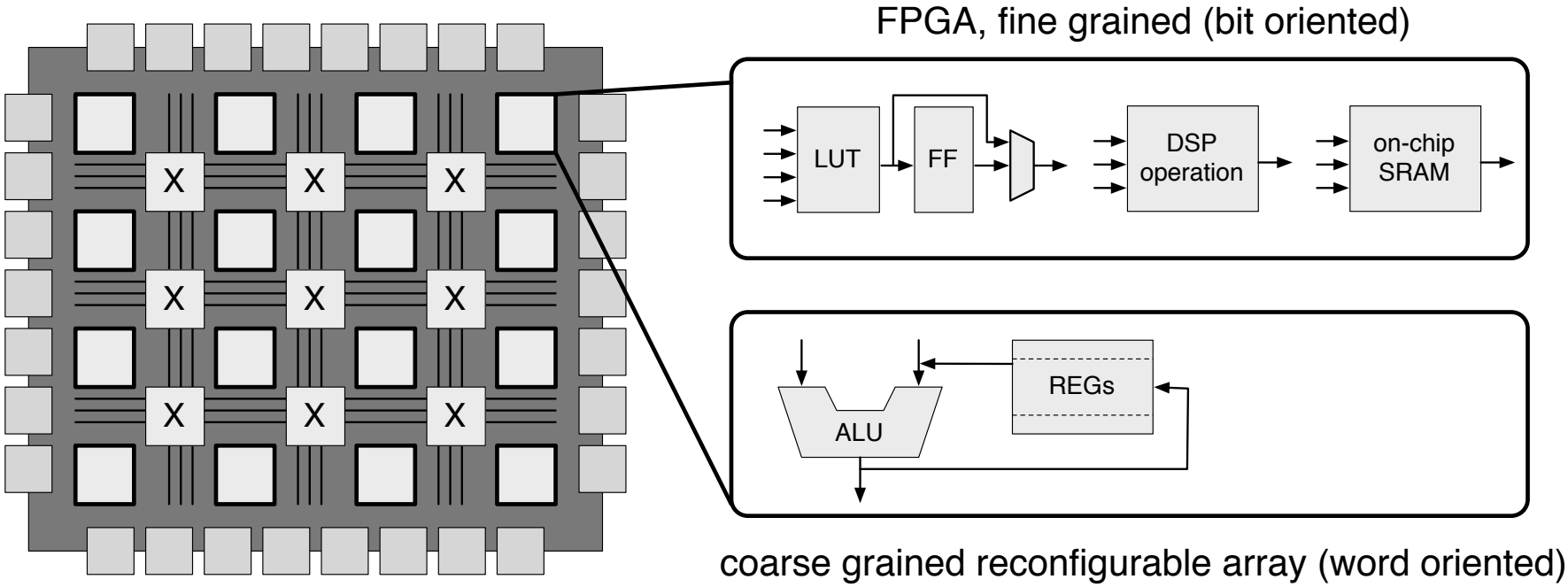
Custom Computing – Example

- bioinformatics: substring search in genome sequences



Custom Computing Technology

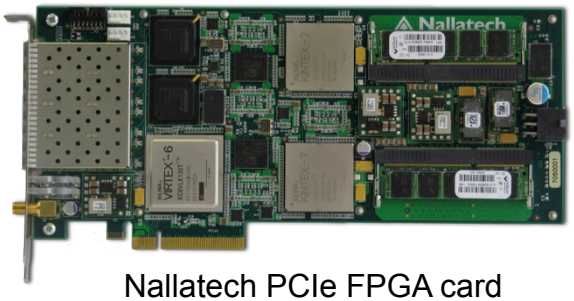
- reconfigurable hardware architecture
  - software programmable processing blocks and
  - software programmable interconnect
  - massively parallel



Maturing Custom Computing Systems

- experimental academic systems
  - proof-of-concept demonstrators
- PCI-attached FPGA acceleration cards
  - most widespread platforms
- custom computing servers
  - Cray, SGI, SRC, Convey, Maxeler, XtremeData, ...
  - targeted at HPC
  - tight integration of CPU and FPGAs
  - domain-specific integrated tool flows

TKDM, in-house development at ETH



Nallatech PCIe FPGA card

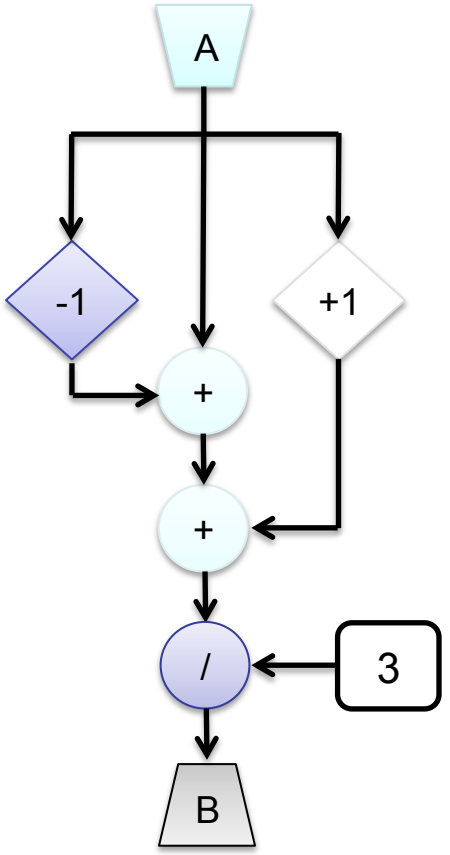
Maxeler MaxNode

Example for an Integrated Custom Computing Solution

- Maxeler dataflow computing
  - tightly integrated hard- and software
  - high-level Java-based specification
  - FPGA internals and tools hidden from developer
  - suitable for streaming applications

b[i] = (a[i-2]+a[i-1]+a[i]) / 3

```
public class Mav_kernel extends Kernel {
  public Mav_kernel (KernelParameters parameters) {
    super(parameters);
    HWVar x = io.input("A", hwFloat(8, 24));
    HWVar prev = stream.offset(x, -1);
    HWVar next = stream.offset(x,  1);
    HWVar sum = prev + x + next;
    HWVar result = sum / 3;
    io.output("B",  result, hwFloat(8, 24));
  }
}
```

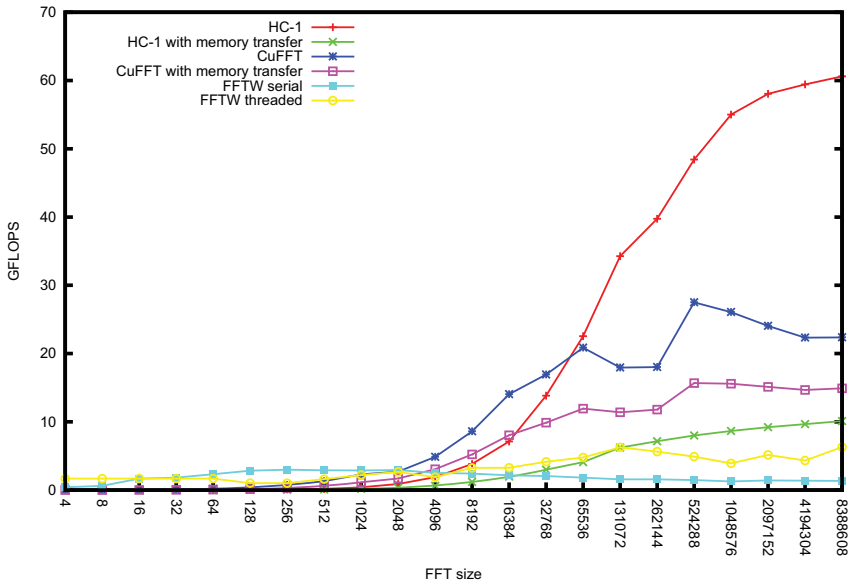


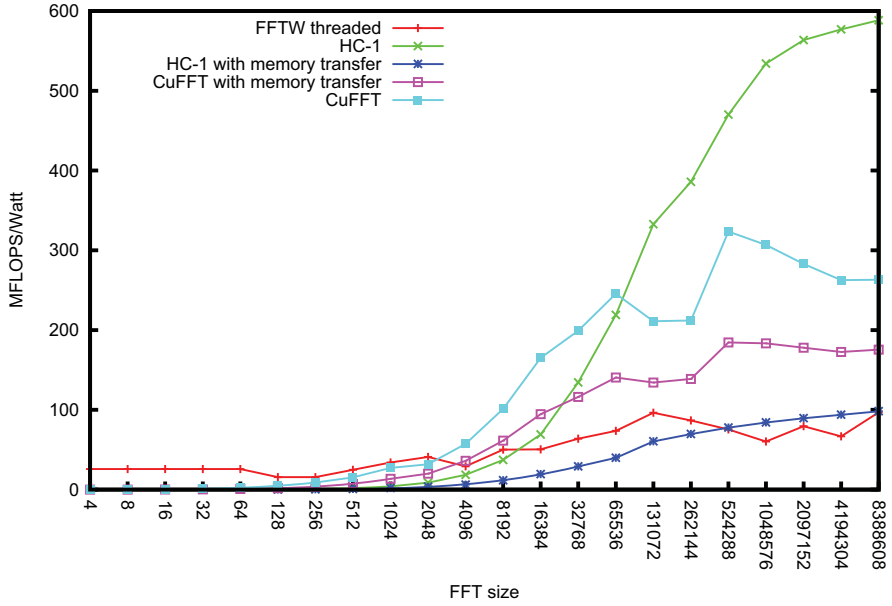
Outline

- history and trends in massively parallel computer architectures
- **comparative case studies**
- simplifying the use of massively parallel architectures
- conclusions and outlook

Case Study 1: FFT Kernel

- performance and energy efficiency for 1D complex in-place FFT
  - CPU (FFTW), GPU (CuFFT), FPGA (Convey HC-1 library)
  - scaling: data size O(N), computation O(N*log(N))
  - significant difference when considering time/energy for copying data to/ from accelerator

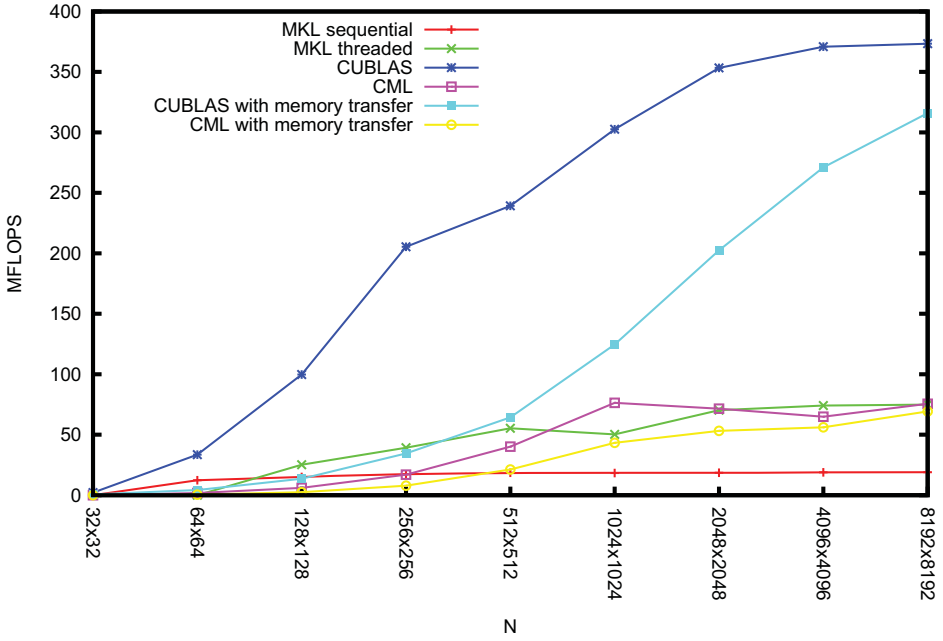


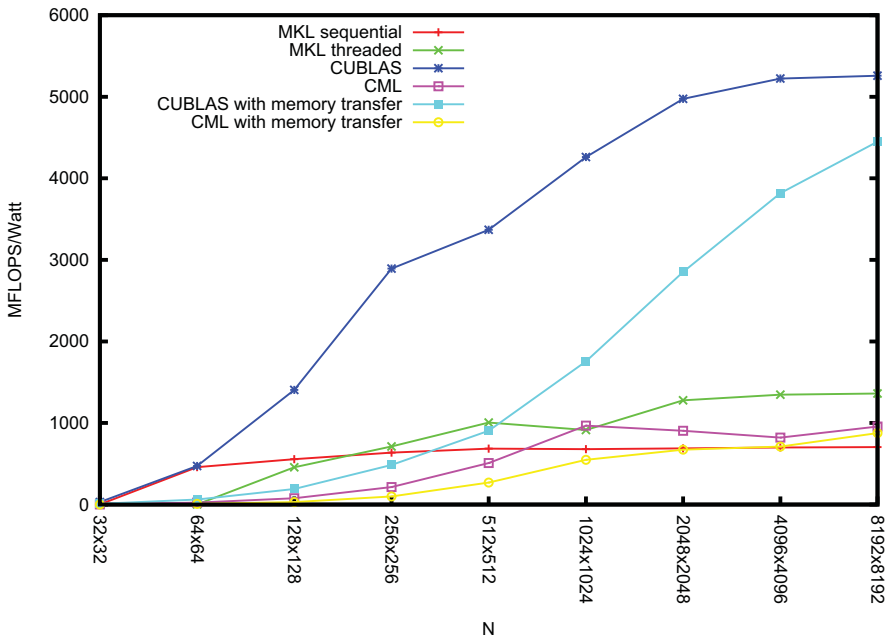


B. Betkaoui, D. B. Thomas, and W. Luk. Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing. In Int. Conf. on Field Programmable Technology (ICFPT), pages 94–101. IEEE. 2010.

Case Study 2: Matrix Multiplication Kernel

- performance and energy efficiency for dense matrix multiplication (DGEMM)
  - CPU (Intel MKL), GPU (CUBLAS), FPGA (Convey math library)
  - scaling: data size O(N^2), computation ~ O(n^3)
  - modest difference when considering time/energy for copying data to/from accelerator





B. Betkaoui, D. B. Thomas, and W. Luk. Comparing performance and energy efficiency of FPGAs and GPUs for high productivity computing. In Int. Conf. on Field Programmable Technology (ICFPT), pages 94–101. IEEE. 2010.

Case Study 3: Computational Finance

- numerical integration for in option pricing (Black Scholes)
- comparison of FPGA, GPU and CPU for 1D integration
  - results are more differentiated
  - GPUs performs well in terms of speed, but have low energy efficiency
  - FPGA is old technology, today's FPGAs would perform even much better

	FPGA		GPU			CPU
	Virtex-4 xc4vlx160		Geforce 8600GT	Tesla C1060		Xeon W3505
Technology	90nm		80nm	65nm		45nm
Release date	Sep 2004		Apr 2007	Sep 2008		Mar 2009
Arithmetic	single	double	single	single	double	double
Clock Rate	100MHz	81.9MHz	1.35GHz	1.3GHz	1.3GHz	3.6GHz
Replicated cores	3	1	-	-	-	-
Processing Speed (M values/sec)	300.0	81.9	114.4	546.5	288.7	65.3
Time for 10^9 values (s)	3.3	12.21	8.74	1.83	3.46	15.31
Acceleration (replicated cores)	**4.59x**	**1.25x**	**1.75x**	**8.37x**	**4.42x**	**1x**
APCC for 10^9 values (W)	4.18	3.3	40.55	102.00	99.00	23.60
AECC for 10^9 values (J)	13.93	40.29	354.42	186.64	342.92	361.30
Normalized energy efficiency	**25.93x**	**8.97x**	**1.02x**	**1.94x**	**1.05x**	**1x**

A. H. T. Tse, D. Thomas, and W. Luk. Design exploration of quadrature methods in option pricing. In IEEE Trans. on Very Large Scale Integration (VLSI) Systems, volume 20, pages 818–826. IEEE, May 2011.

Critical thoughts on Comparative Case Studies

- comparative studies are frequently biased
  - comparison of kernels instead of complete applications (neglect overheads)
  - use of suboptimal code for baseline implementation used for comparison
  - use of unrealistically small or large problem sizes
- not necessarily bad faith
  - different research cultures with different values that define good and interesting research
  - "acceleration experts" jump on hot problems without understanding it sufficiently
- so far, no optimal massively parallel architecture has emerged
  - metrics for success vary: performance, energy/performance, rack space, total cost of ownership, software development cost, tool support, ...
  - all architectures have their sweet spot

Outline

- history and trends in massively parallel computer architectures
- comparative case studies
- **acceleration of algorithms working on regular grids (computational nanophotonics)**
- conclusions and outlook

Outline

- history and trends in massively parallel computer architectures
- comparative case studies
- **simplifying the use of massively parallel architectures**
- conclusions and outlook

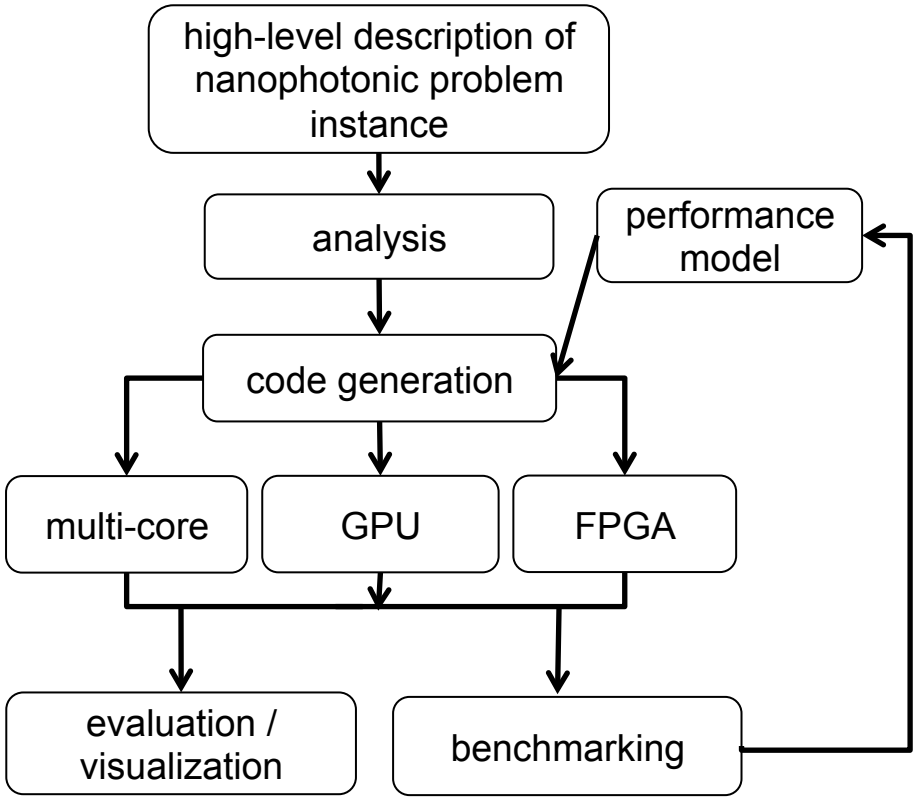
Simplifying the Use of Massively Parallel Architectures

- goal: framework for automated compilation and execution of stencil codes on clusters with massively parallel accelerator architectures
  - collaboration with Jens Förstner, University of Paderborn (theoretical physicist working in computational nanophotonics)

- domain-specific approach
  - focus on iterative algorithms that work on regular grids (e.g. finite difference methods)
  - provide domain specific language allowing scientist to specify problem at a high level
  - create automated tool flow for generating optimized implementations for different massively parallel architectures (cluster with CPUs, FPGAs and/or GPUs)
  - validation with applications from computational nanophotonics

Overview of the envisioned Framework

```
Ey[ix][iy] = ca * Ey[ix][iy] +
             cb * (Hz[ix+1][iy] - Hz[ix][iy]);
```



high-level description

specified by application experts, e.g. physicist

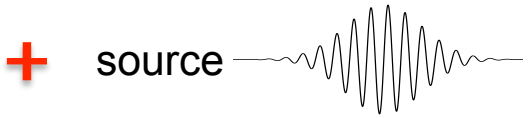
tool-flow

developed by computer architecture and programming expert

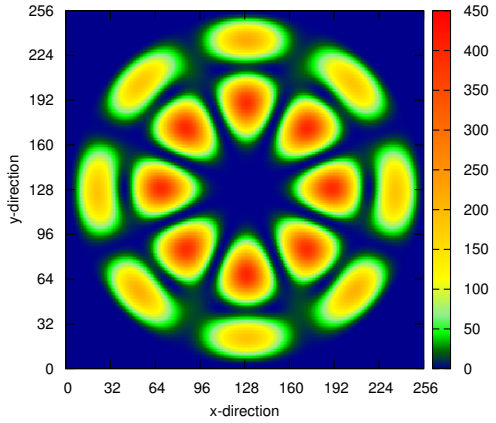
Case Study: Computational Nanophotonics

- test case: microdisk cavity in perfect metallic environment
  - well studied nanophotonic device
  - point-like time-dependent source (optical dipole)
  - known analytic solution (whispering gallery modes)

experimental setup:
microdisk cavity



result: energy density

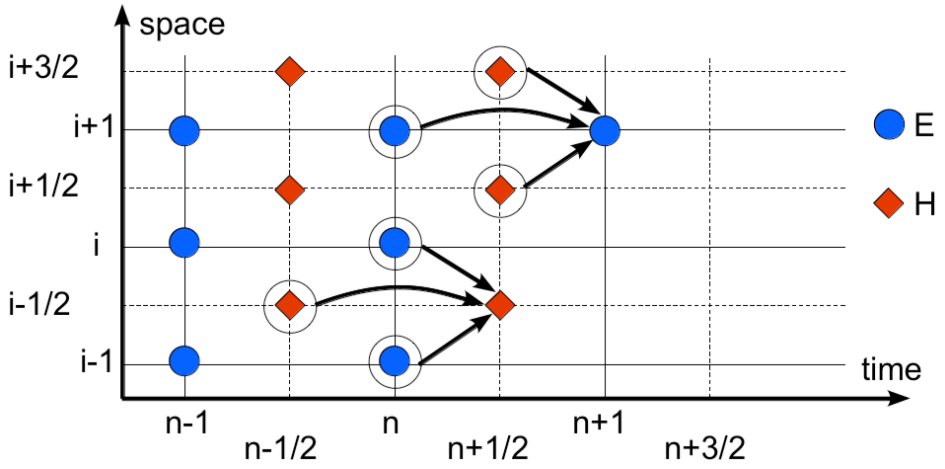


Finite Difference Time Domain Method (FDTD)

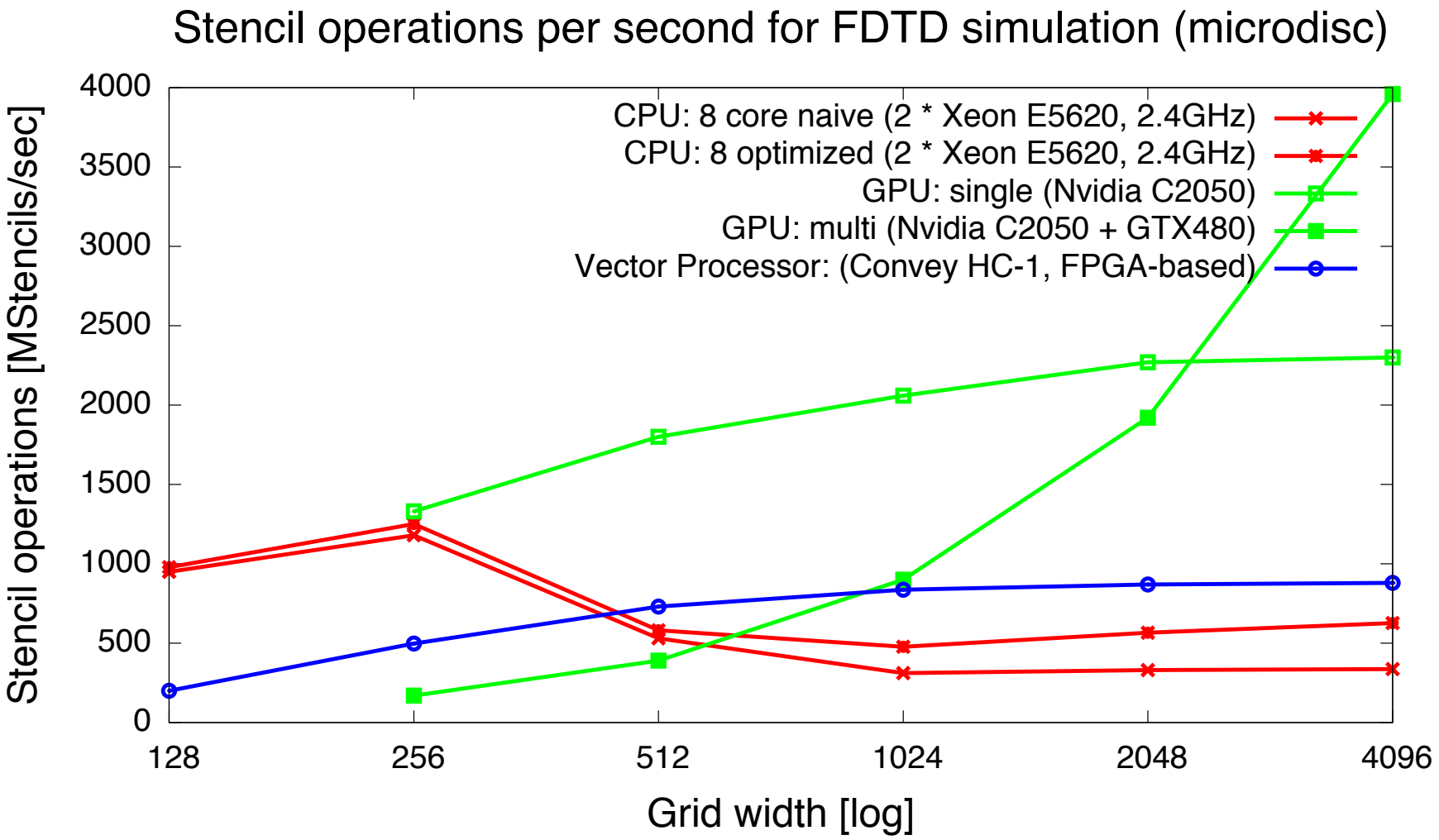
- numerical method for solving Maxwell's equations
- iterative algorithm, computes propagation of fields for fixed time step
- stencil computation on regular grid
  - same operations for each grid point
  - fixed local data access pattern
  - simple arithmetic
- difficult to achieve high performance
  - hardly any data reuse
  - few operations per data

$$
\begin{aligned}
E_x[i] &= ca \cdot E_x[i] + cb \cdot (H_z[i] - H_z[i - dy]) && (1)\\
E_y[i] &= ca \cdot E_y[i] + cb \cdot (H_z[i - dx] - H_z[i]) && (2)\\
H_z[i] &= da \cdot H_z[i] + db \cdot && (3)\\
&\quad \cdot (E_x[i + dy] - E_x[i] + E_y[i] - E_y[i + dx])
\end{aligned}
$$

FDTD update equations
(for one time step)



Results

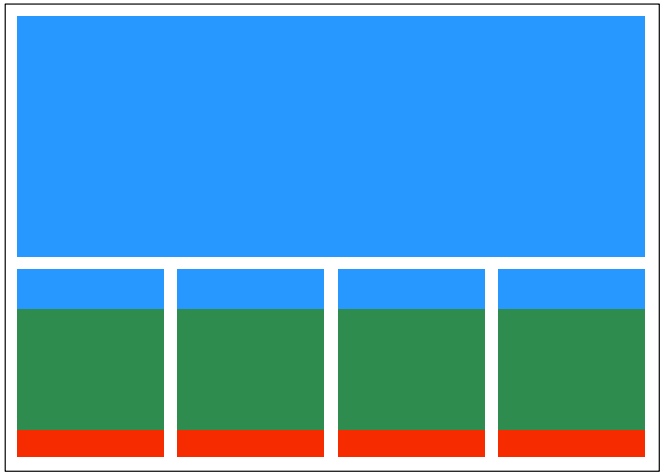


working set size for grid size 512 * 512:
(512*512) * 3 fields * 2 (double buffering) * 4 (double precision) = 6MB

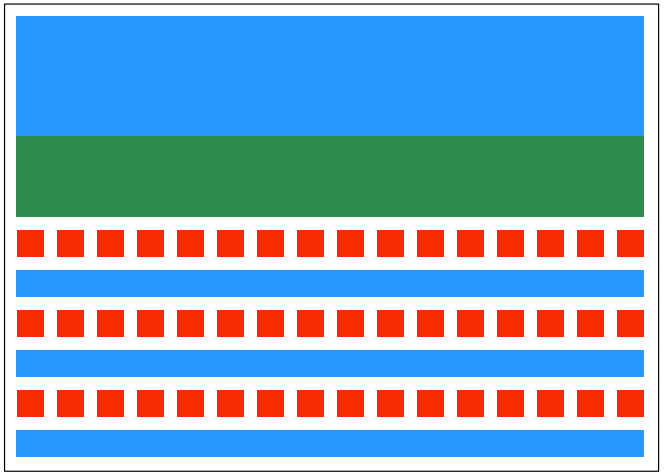
Outline

- history and trends in massively parallel computer architectures
- comparative case studies
- simplifying the use of massively parallel architectures
- **conclusions and outlook**

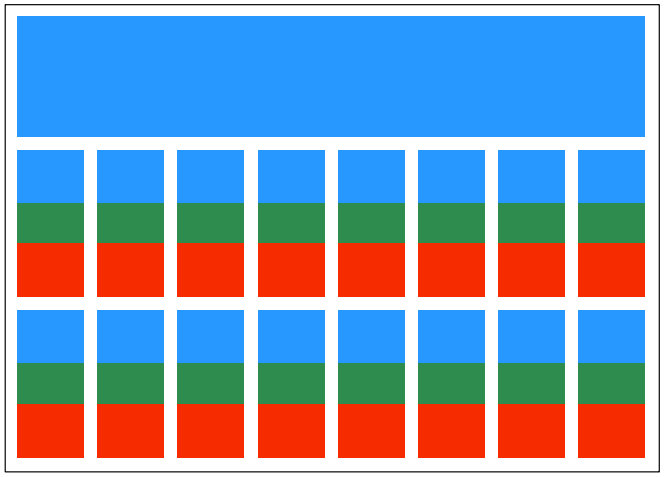
Architecture / Application Sweet Spots



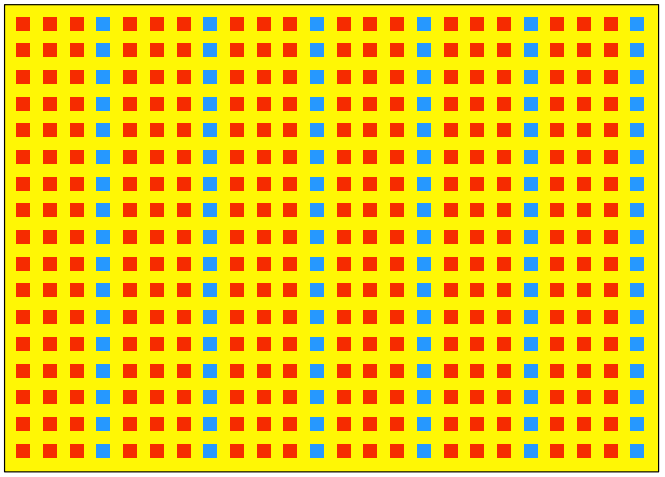
multi-core CPU



GPU



many-core CPU



FPGA

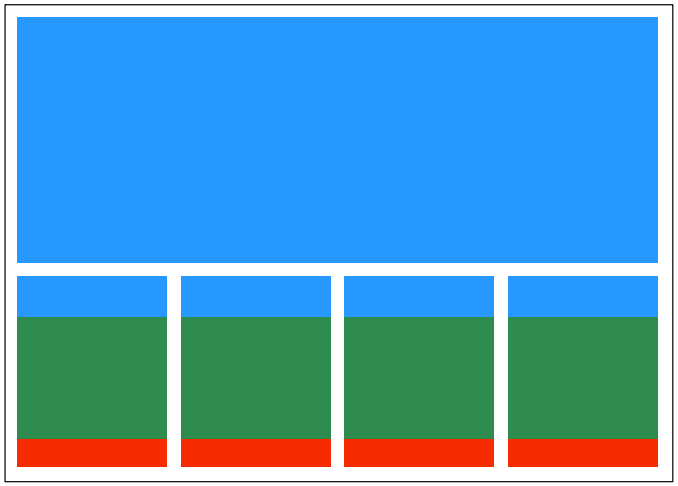
computational unit

execution controller

interconnection network

on-chip memory

Architecture / Application Sweet Spot: Multi-Core

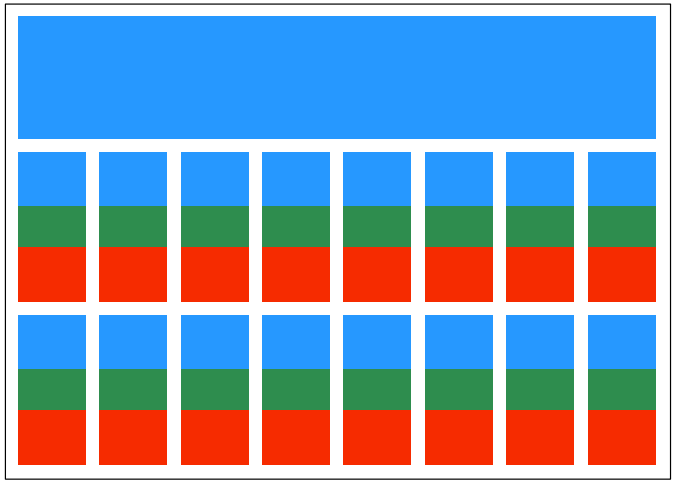


multi-core CPU

multi-cores are suitable for applications that

- scale only moderately (benefit from high single-core performance)
- have threads with different functionality (MIMD)
- are control flow dominated
- rely on shared memory
- have irregular memory access patterns (benefit from caches)
- need access to operating system functions

Architecture / Application Sweet Spot: Many-Core

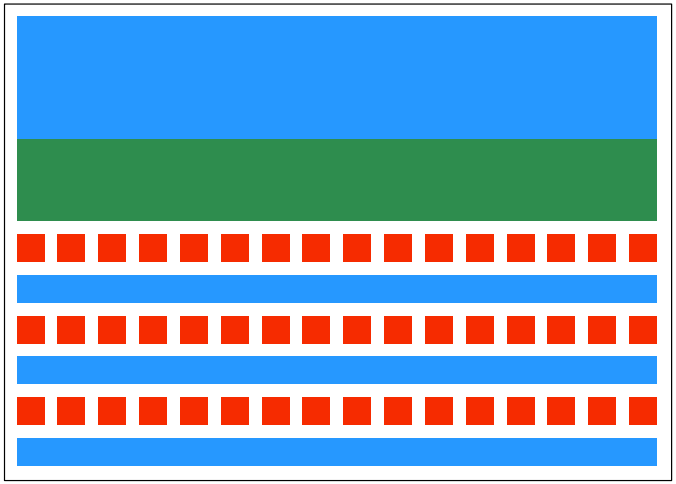


many-core CPU

many-cores are suitable for applications that

- scale well, but threads have different control flows
- compute mostly on private data but may also access shared data
- need access to operating system functions

Architecture / Application Sweet Spot: GPU

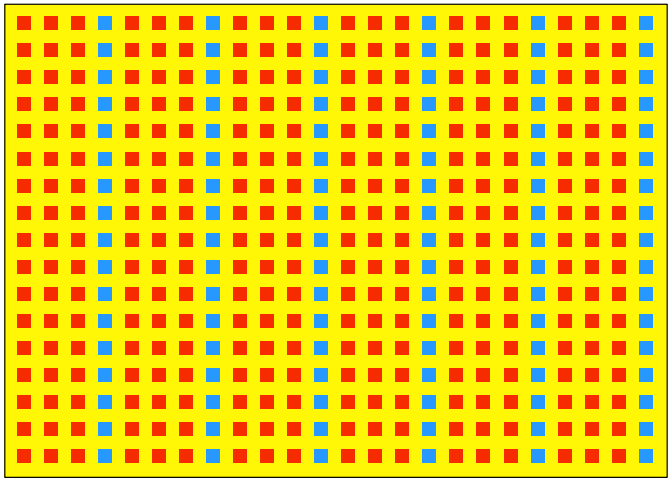


GPU

GPUs are suitable for applications that

- are data parallel
- have threads with identical control flow
- have regular memory access patterns
- work on small, static working sets (no dynamic memory allocation)
- use floating-point arithmetic (preferably single precision)
- do not require operating system calls

Architecture / Application Sweet Spot: FPGA



FPGA

FPGAs are suitable for applications that

- can be expressed as processing pipelines
- process streams of data
- use unconventional arithmetic operations
- use non standard data formats

Conclusions and Outlook

- technological and economical reasons caused CPU performance and efficiency to stagnate
  - field of computer architecture has been revived
  - many new and unconventional ideas are currently explored
  - it seems clear that the future is massively parallel ...
  - ... but there are countless open questions

- opportunities and challenges for computational sciences
  - unprecedented levels of performance become affordable
  - but obtaining high performance comes at a price
  - programming becomes increasingly difficult, the times where computer architecture could be considered a black-box are gone

- many research opportunities for making these new architecture easier to use by non computer scientists

Questions and Contact

- Questions?

- Contact information

Jun.-Prof. Dr. Christian Plessl
christian.plessl@uni-paderborn.de

University of Paderborn
Department of Computer Science