# The Importance of Software in High-Performance Computing

TECHNISCHE UNIVERSITÄT DARMSTADT

Christian Bischof
FG Scientific Computing
Hochschulrechenzentrum
TU Darmstadt
christian.bischof@tu-darmstadt.de



SCIENTIFIC COMPUTING

# WHAT IS HIGH-PERFORMANCE COMPUTING ALL ABOUT?

# Exascale

TECHNISCHE UNIVERSITÄT DARMSTADT

- Frankfurter Allgemeine Zeitung on June 3rd, 2012: „The Yearning of researchers for the exaflops“ (= 10^18 floating point operations per second).
  - „The computing power of supercomputers keeps on rising. And science counts on the continuation of this trend for the future“.

- Putting Exascale in Perspective
  - The human body contains about 10^13 cells, that is one millionth of 10^18 (http://en.wikipedia.org/wiki/Human_microbiome ).
  - Or a million humans contain as many cells as an Exaflop computer performs operation per second.
  - http://htwins.net/scale2/ to probe deeper.

# Important drivers for HPC at TU (but not the only ones)

TECHNISCHE UNIVERSITÄT DARMSTADT

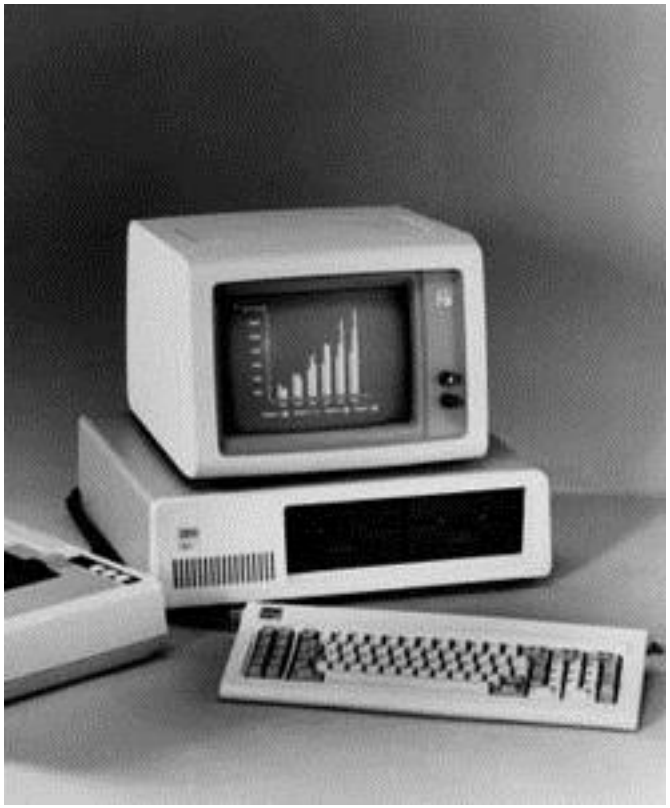
- Graduate School of Computational Engineering, http://www.graduate-school-ce.de/
  - Focus on simulation science in engineering, with recognition of the importance of parallel algorithms and software.
  - See also „forschen“, Nr. 2/2011, (in German) http://www.tu-darmstadt.de/vorbeischauen/publikationen/forschung/archiv/aktuellethemaforschung_3136.de.jsp
- Darmstadt Graduate School of Energy Science and Engineering, http://www.ese.tu-darmstadt.de/graduate_school_ese/gsc_welcome/willkommen_1.en.jsp
  - Focus on transition from non-renewable to renewable and environmentally friendly energy resources
- CSI = Cluster Smart Interfaces, http://www.csi.tu-darmstadt.de/
  - Focus on understanding and designing fluid boundaries.
  - See also „forschen“, Nr. 2/2009, (in German) http://www.tu-darmstadt.de/vorbeischauen/publikationen/forschung/archiv/aktuellethemaforschung_1344.de.jsp

# COMPUTING SPEED

# A look back: The IBM PC (1981)

TECHNISCHE UNIVERSITÄT DARMSTADT

PC = **Personal** Computer

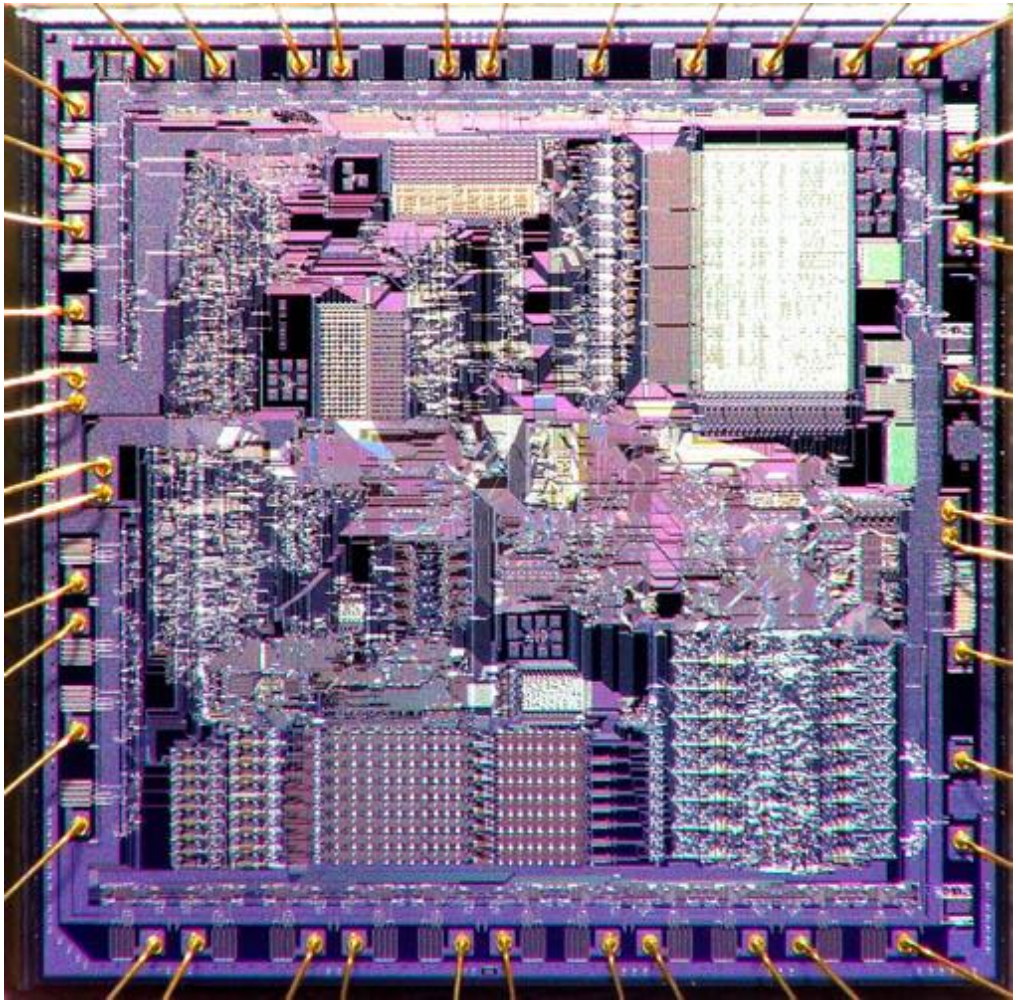


http://dl.maximumpc.com/galleries/25oldpcs/IBM_PC_XT_01_full.jpg

Before, computers were something special, hidden far away and cared for by high priests called operators

http://images.yourdictionary.com/images/computer/_IBMMFRM.GIF

# The Intel 8088 chip inside the IBM PC



- 29000 transistors
- Clocked at 4.77 Mhz
- At most 256 kByte RAM

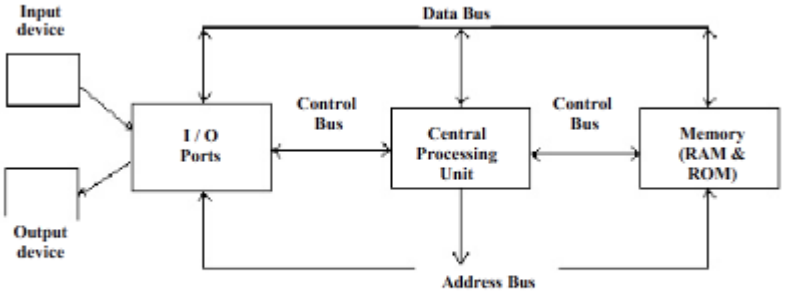



**Fig. 1 Block Diagram of a simple Computer or a Microcomputer.**

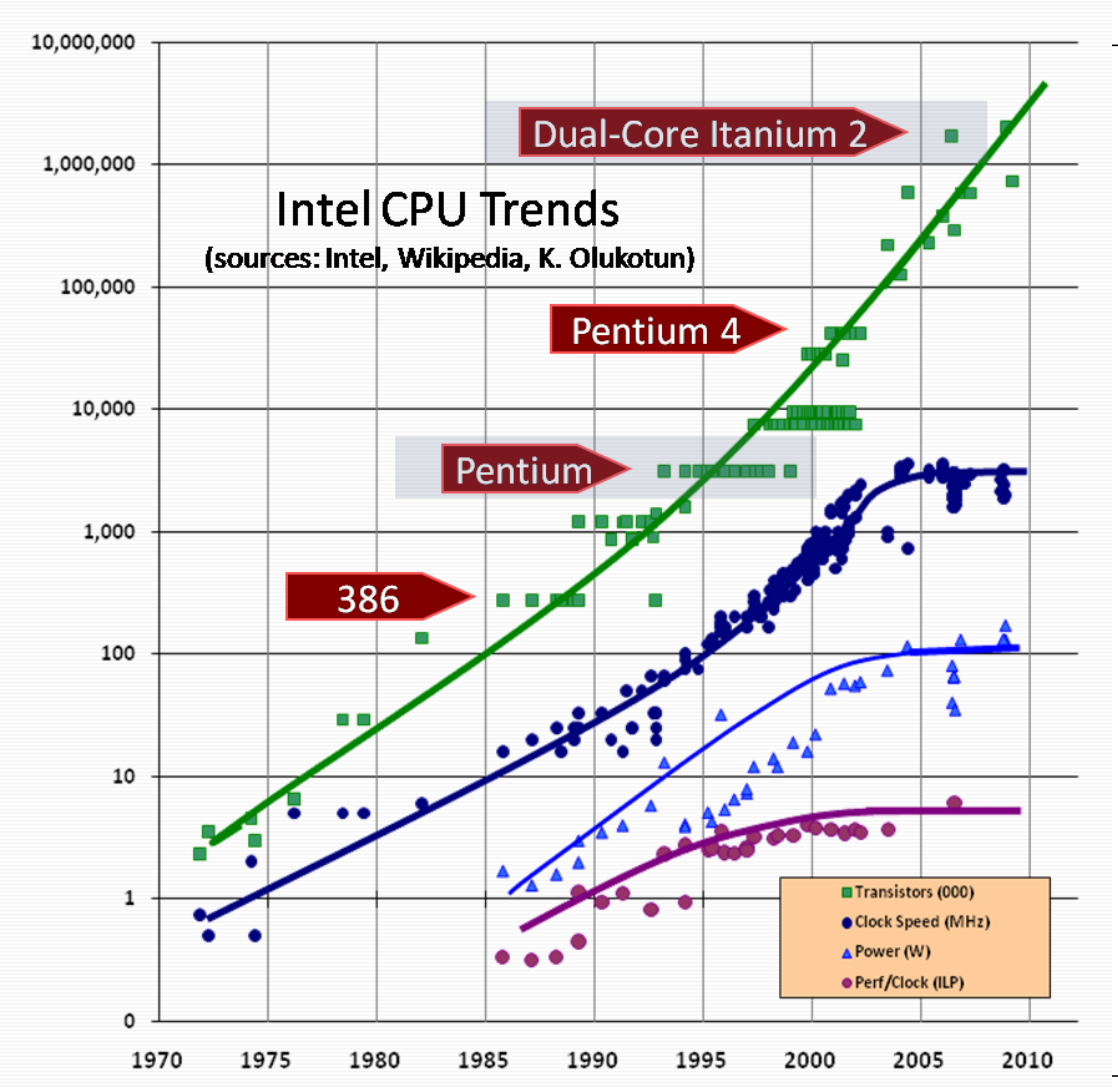
http://www.cpu-galaxy.at/cpu/intel%20cpu/8088-80286/Intel%208088%20section-Dateien/intel%208088%20Core.jpg

# The first supercomputer: Cray 1 (1976)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Cray-1 @ Deutsches Museum, Munich
  http://commons.wikimedia.org/wiki/File:Cray-1-deutsches-museum.jpg
- First machine for Los Alamos (for nuclear stewardship)
- Weight of 5,5 tons
- Clocked at 80 MHz
- 8 Mbyte RAM
- 8,8 Mio $
- A special computer for science
- In the basement: Electricity, cool water

# Moore‘s law




Computing speed doubles every 18 months

The reason:

- Number of transistors for the same expenditure doubles every two years
- Clock speed increases

… or at least clock speed used to increase

Source: Herb Sutter
www.gotw.ca/publications/concurrency-ddj.htm

# The world of HPC looks like a great place!

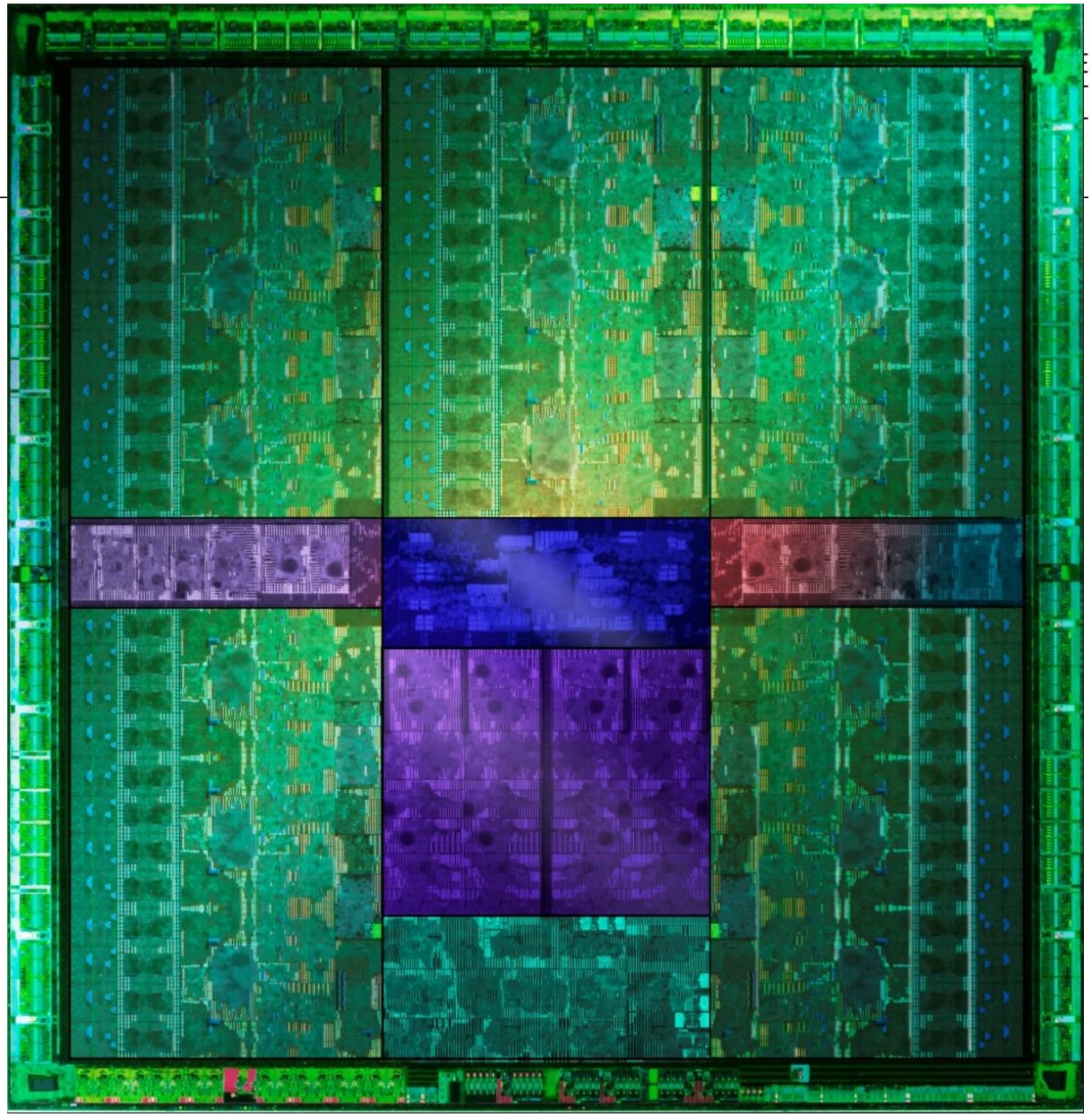
TECHNISCHE UNIVERSITÄT DARMSTADT

- Moore‘s Law: The number of transistors for the same amount of money doubles every 18 months.
  - Unfortunately memory bandwidth doubles only every 6 years.
- Processors in many „flavors“, all of them are multi-/manycore systems
- Keep in mind that the power consumption (and heat produced) skales linearly with the number of transistors, but quadratically with the clock rate.

# NVIDIA Kepler Chip Layout

- 7,1 billion transistors
- Clocked at 1,3 GHz
- 1536 cores

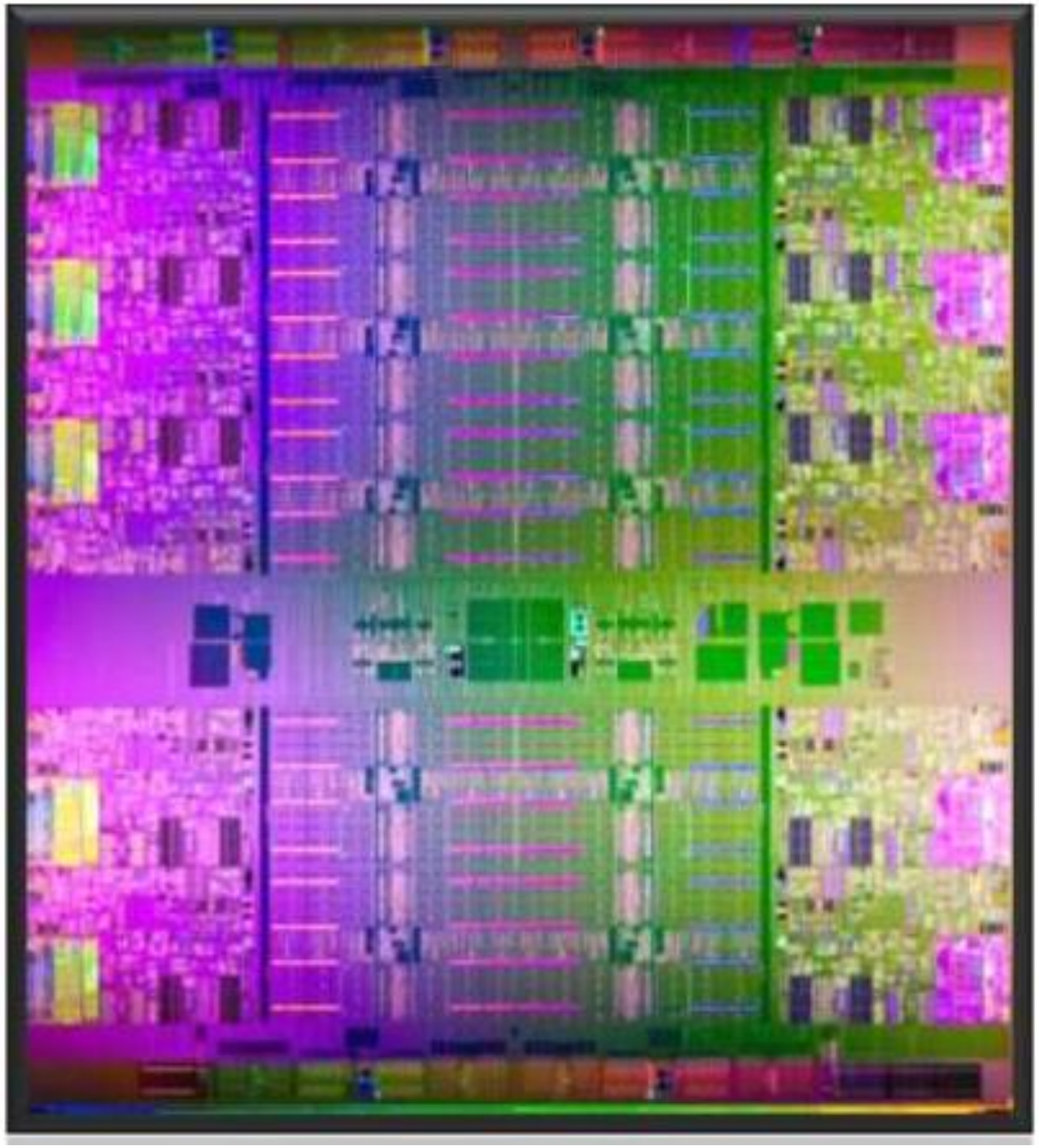
http://www.nvidia.de/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf



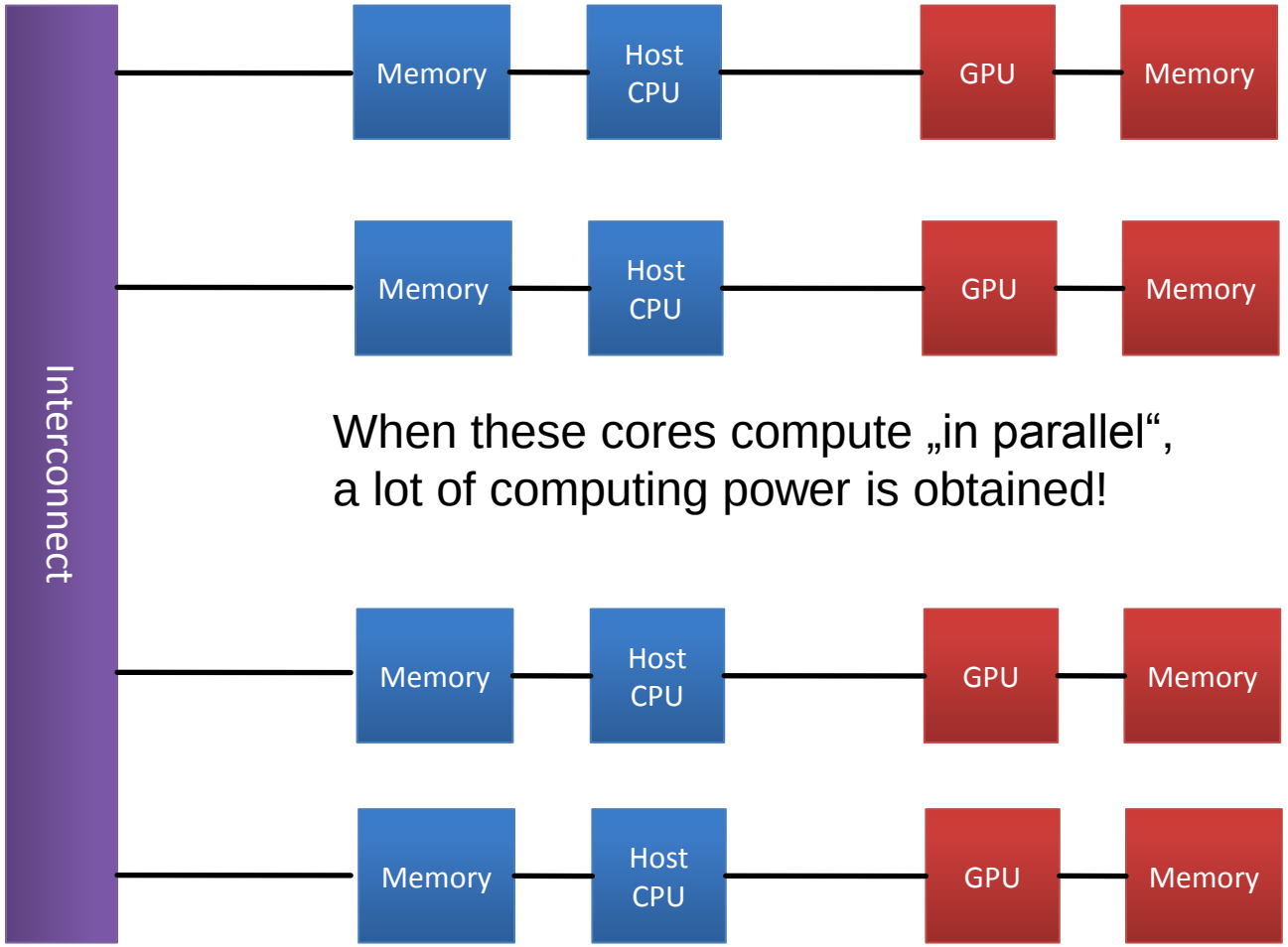
# Intel Westmere-Ex Chip Layout

- 2,9 billion transistors
- Clocked at 2,4 GHz
- 10 cores (each capable of 8-way vector instructions)

http://www.heise.de/newsticker/meldung/Intels-neues-Prozessor-Flaggschiff-Westmere-EX-1222278.html



# HPC from standard components



# The fastest computers

TOP500 NOVEMBER 2012

TECHNISCHE UNIVERSITÄT DARMSTADT

| | NAME | SPECS | SITE | COUNTRY | CORES | RMAX PFLOP/S | POWER MW |
|---|---|---|---|---|---|---|---|
| 1 | TITAN | Cray XK7, Operon 6274 16C 2.2 GHz + Nvidia Kepler GPU, Custom interconnect | DOE/OS/ORNL | USA | 560,640 | **17.6** | 8.3 |
| 2 | SEQUOIA | IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect | DOE/NNSA/LLNL | USA | 1,572,864 | **16.3** | 7.9 |
| 3 | K COMPUTER | Fujitsu SPARC64 VIIIfx 2.0GHz, Custom interconnect | RIKEN AICS | Japan | 705,024 | **10.5** | 12.7 |
| 4 | MIRA | IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect | DOE/OS/ANL | USA | 786,432 | **8.16** | 3.95 |
| 5 | JUQUEEN | IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect | Forschungszentrum Jülich | Germany | 393,216 | **4.14** | 1.97 |

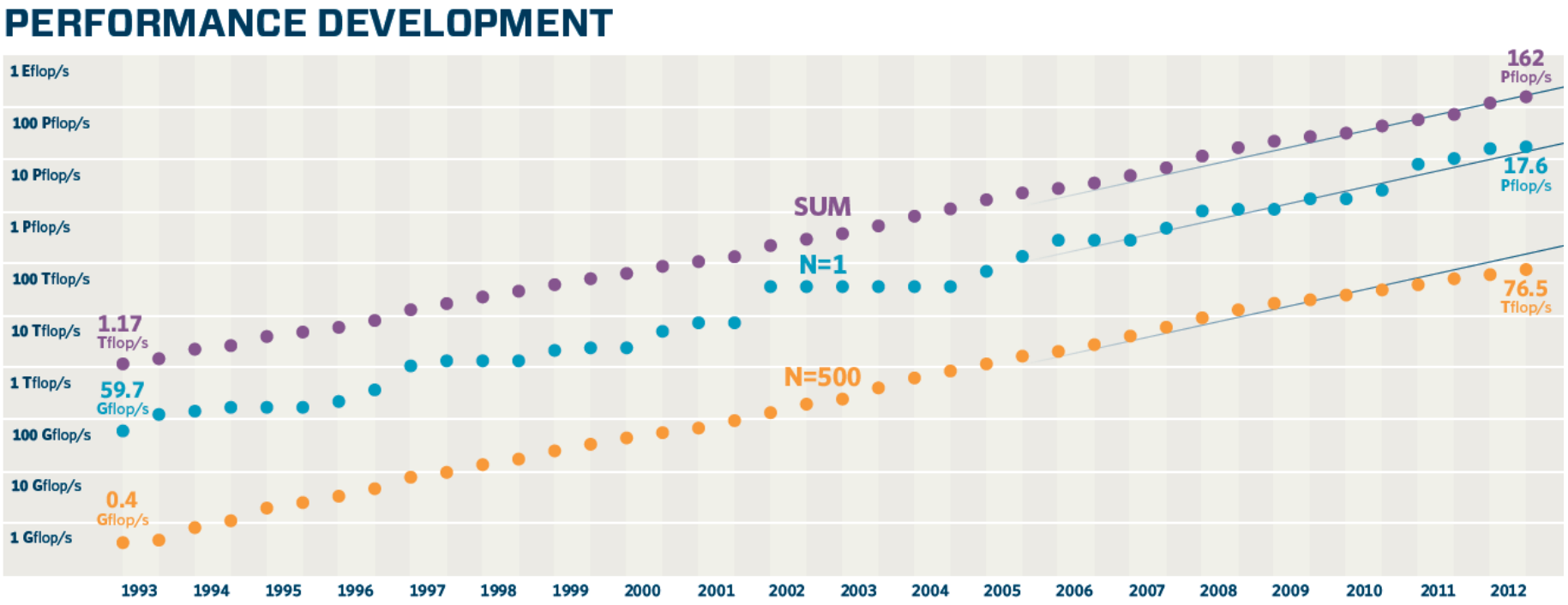
See http://s.top500.org/static/lists/2012/11/TOP500_201211_Poster.pdf


http://www.fujitsu.com/img/KCOM/top/fujitsu-kcomputer-2012-e.jpg

# Measuring top speed

TECHNISCHE UNIVERSITÄT DARMSTADT

HPC Computers are usually ranked with the so-called Linpack-Benchmark, i.e. the solution of a linear equation system Ax=b with the Gauß-Algorithm, see www.top500.org






SCIENTIFIC COMPUTING

# The Gauß-Algorithm, from Bronstein und Semendjajew: Taschenbuch der Mathematik, 1957 (19th edition 1979)

TECHNISCHE UNIVERSITÄT DARMSTADT

*7.1.2.1.1. Direkte Methoden (Gaußsche Elimination)*

(1) **Das einfache Gauß-Verfahren:** Das bekannte Eliminationsverfahren (vgl. 2.4.4.3.3.) besteht nach der Umformung in einen Algorithmus aus zwei zyklisch gestalteten Teilprozeduren.

*Transformation von* A *auf Dreiecksgestalt*:

1. Setze $k = 1$.
2. Prüfe, ob $a_{kk} \neq 0$ ist.
3. Wenn ja, dann wird die $k$-te Zeile *Arbeitszeile* und $a_{kk}$ *Pivotelement*. Wenn nicht, so vertauschen wir die $k$-te mit einer $l$-ten Zeile ($l > k$), in welcher $a_{lk} \neq 0$ ist.
4. Für $i = k + 1, k + 2, \ldots, n$ berechnen wir neue Matrixelemente, die wir wie die alten bezeichen wollen, nach folgender Vorschrift: Bilde $q_i = -\frac{a_{ik}}{a_{kk}}$;

$$a_{ij} := 0 \quad \text{für} \quad j = k,$$

$$a_{ij} := a_{ij} + q_i a_{kj} \quad \text{für} \quad j \neq k,$$

und analog bilden wir neue rechte Seiten nach

$$b_i := b_i + q_i b_k.$$

5. Wir erhöhen $k$ um eins, d. h. $k := k + 1$, und beginnen erneut mit dem 2. Schritt, sofern $k$ noch kleiner als $n - 1$ ist. Anderenfalls sind wir fertig und haben eine obere Dreiecksmatrix erhalten

$$\mathrm{A}' = \begin{pmatrix} a'_{11} & a'_{12} & \ldots & a'_{1n} \\ 0 & a'_{22} & \ldots & a'_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & \ldots & a'_{nn} \end{pmatrix}.$$

*Berechnung des Lösungsvektors* $\mathbf{x}^{\mathrm{T}} = (x_1, x_2, \ldots, x_n)$:

1. Berechne $x_n = \frac{b'_n}{a'_{nn}}$.
2. Berechne für $i = (n - 1), (n - 2), \ldots, 1$

$$x_i = \frac{1}{a'_{ii}} \left( b'_i - \sum_{j=1}^{n-i} a'_{ii+j} x_{i+j} \right).$$

An optimized implementation of the Linpack benchmark may contain 75,000 LOC (Ruud van der Pas, Oracle, pers. Kommunikation)

The organisation of memory accesses makes life complicated!

The memory subsystem and system interconnect greatly impact Linpack performance:

- Titan: 63% of peak
- K computer: 94%
- JUQUEEN: 82%
- SuperMUC: 91%
- Tianhe-1A: 55%

# Good news and bad news

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Good news:** In light of the size and complexity of todays high-end computing systems it is astounding that we can get the performance documented by the Linpack benchmark.
- **Bad news:** It does not do anything for science!
  - I know of no application that solves equation systems that big with the Gauß Algorithm.
  - O(n*n*n) flops with O(n*n) data allows masking of slow memories.
  - On a fast Fourier transform, that requires O(n log n) flops on O(n) data performance looks very different!
- „Green IT“ in the sense of optimizing power per flop for the Linpack benchmark (www.green500.org) is not relevant for most real computing.
- Challenge: **Programming high-performance computers for scientifically relevant applications.**

# HPC Programming Standards

TECHNISCHE UNIVERSITÄT DARMSTADT

Parallel computing in a distributed memory environment

- MPI (Message-Passing Interface)

Multithreading in a shared memory environment:

- OpenMP
- Pthreads

Vectorization:

- CUDA (Nvidia proprietary)
- OpenCL (open specification)
- OpenACC (sort of open)

# OpenMP Example for SAXPY



- SAXPY: Vector operation y(:) := a*x(:) + y(:)

```
#pragma omp parallel for
  for ( i = 0; i < n; i++ )
    y[i] = a * x[i] + y[i];
```

- Different iterations are automatically assigned to different cores.
- Relatively high level of abstraction, with further directives to address thread placement and binding, as well as scheduling.
- Task- as well as loop parallelism
- Bindings for C/C++/Fortran/Java

# OpenCL example for SAXPY

## Close to a hardware model, low level of abstraction

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#include <stdio.h>
#include <CL/cl.h>
const char* source[] = {
  "__kernel void saxpy_opencl(int n, float a, __global float*
     x, __global float* y)",
  "{",
  "  int i = get_global_id(0);",
  "  if( i < n ){",
  "    y[i] = a * x[i] + y[i];",
  "  }",
  "}"
};
int main(int argc, char* argv[]) {
  int n = 10240;  float a = 2.0;
  float* h_x, *h_y; // Pointer to CPU memory
  h_x = (float*) malloc(n * sizeof(float));
  h_y = (float*) malloc(n * sizeof(float));
  // Initialize h_x and h_y
  for(int i=0; i<n; ++i){
    h_x[i]=i; h_y[i]=5.0*i-1.0;
  }
  // Get an OpenCL platform
  cl_platform_id platform;
  clGetPlatformIDs(1,&platform, NULL);
  // Create context
  cl_device_id device;
  clGetDeviceIDs(platform, CL_DEVICE_TYPE_GPU, 1, &device,
     NULL);
  cl_context context = clCreateContext(0, 1, &device, NULL,
     NULL, NULL);
  // Create a command-queue on the GPU device
  cl_command_queue queue = clCreateCommandQueue(context,
     device, 0, NULL);
  // Create OpenCL program with source code
  cl_program program = clCreateProgramWithSource(context, 7, source,
     NULL, NULL);
  // Build the program
  clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
  // Allocate memory on device on initialize with host data
  cl_mem d_x = clCreateBuffer(context, CL_MEM_READ_ONLY |
     CL_MEM_COPY_HOST_PTR, n*sizeof(float), h_x, NULL);
  cl_mem d_y = clCreateBuffer(context, CL_MEM_READ_WRITE |
     CL_MEM_COPY_HOST_PTR, n*sizeof(float), h_y, NULL);
  // Create kernel: handle to the compiled OpenCL function
  cl_kernel saxpy_kernel = clCreateKernel(program, "saxpy_opencl",
     NULL);
  // Set kernel arguments
  clSetKernelArg(saxpy_kernel, 0, sizeof(int), &n);
  clSetKernelArg(saxpy_kernel, 1, sizeof(float), &a);
  clSetKernelArg(saxpy_kernel, 2, sizeof(cl_mem), &d_x);
  clSetKernelArg(saxpy_kernel, 3, sizeof(cl_mem), &d_y);
  // Enqueue kernel execution
  size_t threadsPerWG[] = {128};
  size_t threadsTotal[] = {n};
  clEnqueueNDRangeKernel(queue, saxpy_kernel, 1, 0, threadsTotal,
     threadsPerWG, 0,0,0);
  // Copy results from device to host
  clEnqueueReadBuffer(queue, d_y, CL_TRUE, 0, n*sizeof(float), h_y,
     0, NULL, NULL);
  // Cleanup
  clReleaseKernel(saxpy_kernel);
  clReleaseProgram(program);
  clReleaseCommandQueue(queue);
  clReleaseContext(context);
  clReleaseMemObject(d_x); clReleaseMemObject(d_y);
  free(h_x); free(h_y); return 0;
}
```

# Program versus Software

TECHNISCHE UNIVERSITÄT DARMSTADT

- A clever student that writes and debugs an efficient **program** for a particular HPC platform is a real asset.

- **Software** is, in addition
  - Validated
  - Extensible
  - Maintainable
  - Documented

- HPC software needs to be properly designed with a long-term view, and continually cared for.
  - **Software has a much longer life cycle than hardware!**

# THE SOFTWARE GAP

# The software gap

TECHNISCHE UNIVERSITÄT DARMSTADT

- Parallel programming is intellectually challenging.
- The low level of abstraction offered by today‘s prevalent parallel computing standards does not ease the complexity and encourages programming towards a particular architecture.
  - If the code moves onto a newer machine, performance can easily decrease!

- **The software gap**: „Today‘s CSE ecosystem is unbalanced, with a software base that is inadequate to keep pace with evolving hardware and application needs“
  Presidents Information Technology Advisory Committee - Report to the President 2005: *Computational Science: Ensuring America‘s competitiveness*

# Productivity versus Performance

TECHNISCHE UNIVERSITÄT DARMSTADT

- Traditionally, HPC programming has been slanted towards **performance –** heroic programmers getting good performance on „their“ machine.
- In the future, **productivity** is much more important, i.e. how long does it take for a scientist to express his ideas in a code that is
  - correct
  - makes sensible use of HPC resources, and
  - can be maintained over a series of architectures.
- Note: Moore‘s law enabled a great variety of machines, and resource virtualization put many machines at our fingertips - so use the machine that fits best to your problem!

# Two approaches to closing the software gap

TECHNISCHE UNIVERSITÄT DARMSTADT

**1) Investing in HPC software stewardship**:

- Provide „brainware“, i.e. HPC experts that regularly adapt codes to changing HPC environments.
- That is, software, like hardware, is viewed as an infrastructure that is continually cared for.

**2) Increasing the productivity of programming through automation**

> **“The way you get programmer productivity is by eliminating lines of code you have to write.”**
>
> – Steve Jobs, Apple World Wide Developers Conference, Closing Keynote Q&A, 1997

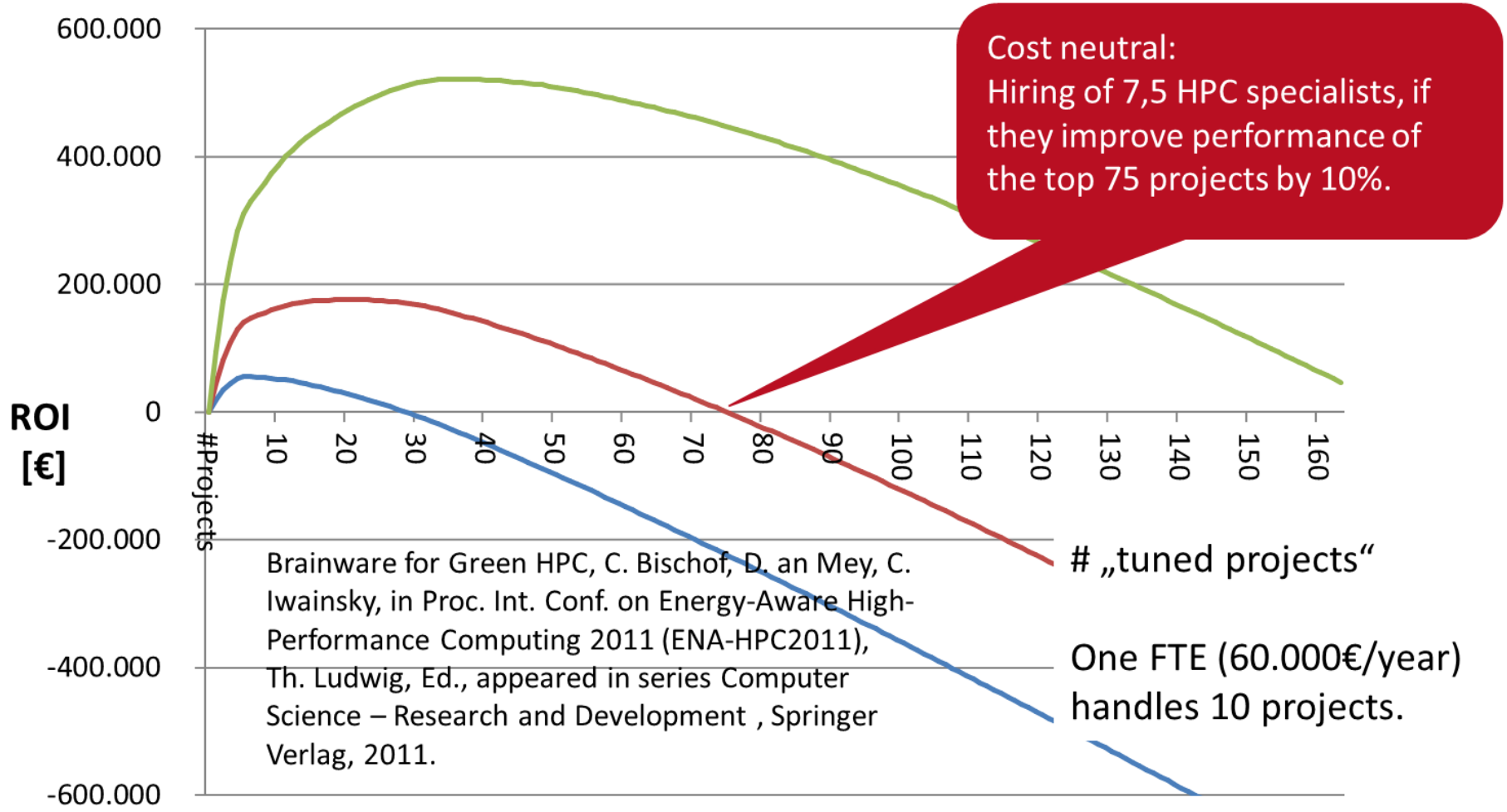
TECHNISCHE UNIVERSITÄT DARMSTADT

# BRAINWARE FOR HPC SOFTWARE STEWARDSHIP



SCIENTIFIC COMPUTING

# Return of Investment of Brainware (5/10/20% Improvement)



# Brainware pays off – the UK CSE program

TECHNISCHE UNIVERSITÄT DARMSTADT

CP2K (materials science) - 15% Speed and 2X Scalability for 18 person months effort. This code uses ~10,000k AUs per month giving ~ 84k £ saving per annum.

CABARET (quasi-geostrophic ocean model) - 2X Speed and 2X Scalability for 2 person months effort. This code uses ~1,000k AUs per month, ~69k £ saving per annum.

DL_POLY_4 (classical MD code) - 7X Speed and 4X Scalability for 12 person months effort. This code uses ~9,000k AUs per month > 1,000k £ saving per year.

GS2 (magnetised plasma simulation code) - representative 10% Speed up for 12 person months effort. The code has 150,000k AUs assigned for its use, a saving of at least £90k is possible.

CASTEP (first principle DFT calculations for the electronic properties of materials) - 2X Speed and 4X Scalability for 12 person months effort. This code uses ~17,000k AUs per month > 1,179k £ saving per year.


http://www.hector.ac.uk/cse/reports/


Slide courtesy of Phil Ridley, NAG

# Brainware also pays off at TU!

TECHNISCHE UNIVERSITÄT DARMSTADT

- Please put to work in your codes what you will learn this week about
  - Tuning, mapping, libraries, compilers
  - Cache, OpenMP, MPI, CUDA optimization
  - Hybrid parallelization
- This way, you are contributing to an efficient and „green“ operation of the TU computing system, which is a major TU investment!
  - As a rule of thumb, total cost of ownership (TCO) for a HPC system is twice the hardware invest!

TECHNISCHE UNIVERSITÄT DARMSTADT

# AUTOMATING SOFTWARE GENERATION



SCIENTIFIC COMPUTING

# Automating code generation

TECHNISCHE UNIVERSITÄT DARMSTADT

- **If we know enough about the problem domain, we can automate code generation.**

**Examples:**

- Automatic Differention (AD)
- Domain-Specific Languages
- Automating Performance Modeling

# Automatic Differentiation (AD)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Given a computer program, AD generates a new program, which applies the chain rule of differential calculus, e.g.,

$$\frac{df(w,v)}{dx} = \frac{\partial f}{\partial w} * \frac{dw}{dx} + \frac{\partial f}{\partial v} * \frac{dv}{dx}$$

  to elementary operations (i.e., $\frac{\partial f}{\partial w}, \frac{\partial f}{\partial v}$ are known).

- Unlike divided differences $\frac{f(x+h) - f(x)}{h}$

  AD does not generate truncation- or cancellation errors.

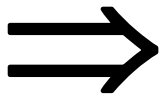
- AD generates a program for the computation of derivative values, not a closed formula for the derivatives.

# The Forward Mode (FM) of AD

TECHNISCHE UNIVERSITÄT DARMSTADT

- Associate a "gradient" $\nabla$ with every program variable and apply derivative rules for elementary operations.

```
t := 1.0
Do i=1 to n step 1
   if (t > 0) then
      t := t*x(i);
   endif
endif
```



```
t := 1.0; ∇ t:= 0.0;
Do i=1 to n step 1
   if (t > 0) then
      ∇ t := x(i)* ∇ t+t* ∇ x(i);
      t := t*x(i);
   endif
endif
```

- the computation of $p$ (directional) derivatives requires gradients of length $p \rightarrow$ many vector linear combinations,
  - e.g., $\nabla x(i) = (0,..,0,1,0,..,0)$, i=1,…,n results in $\nabla t ==$ dt/dx(1:n).
  - e.g., $\nabla x(i) = 1$, $\nabla x(j) = 0$ for $j \neq n$ results in $\nabla t ==$ dt/dx(i).
  - e.g., $\nabla x(i) = z(i)$, i=1,…,n results in $\nabla t ==$ dt/dx(1:n)*z.

# The Reverse Mode (RM) of AD

TECHNISCHE UNIVERSITÄT DARMSTADT

Associate an "adjoint" $\alpha$ with every program variable and apply the following rules to the "inverted" program:
s = f(v,w) results in **$\alpha$_v += ds/dv* $\alpha$_s; $\alpha$_w += ds/dw* $\alpha$_s; $\alpha$_s := 0;**

```
tval(0) := 1.0; tctr := 0;
Do i = 1 to n step 1
    If (tval(tctr)>0) then
        jump(i):= true; tctr = tctr +1;
        tval(tctr) := tval(tctr-1)*x(i);
```

**1. Step: Make program reversible**

- Here by transformation into single assignment form
- jump(1:n) initialized to 'false'

**2. Step: Adjoint computation**

- Initially $\alpha$_tval(tctr) =1.0, all other adjoints are set to zero.
- Upon exit, $\alpha$_x(i)==dt/dx(i)

```
Do i = n downto 1 step -1
    if (jump(i) == true) then
        α_x(i) += tval(tctr-1)*α_tval(tctr);
        α_tval(tctr-1) += x(i)*α_tval(tctr);
        α_tval(tctr) = 0; tctr := tctr – 1;
```

# The Reverse Mode (RM) of AD – now for t = t+x(i)

TECHNISCHE UNIVERSITÄT DARMSTADT

Associate an "adjoint" $\alpha$ with every program variable and apply the following rules to the "inverted" program:
s = f(v,w) results in **$\alpha$_v += ds/dv* $\alpha$_s; $\alpha$_w += ds/dw* $\alpha$_s; $\alpha$_s := 0;**

```
tval(0) := 1.0; tctr := 0;
Do i = 1 to n step 1
    If (tval(tctr)>0) then
        jump(i):= true; tctr = tctr +1;
        tval(tctr) := tval(tctr-1) + x(i);
```

**1. Step: Make program reversible**

- Here by transformation into Single Assignment Form
- jump(1:n) initialized to 'false'

**2. Step: Adjoint computation**

- Initially $\alpha$_tval(tctr) :=1.0, all other adjoints are set to zero.
- Upon exit, $\alpha$_x(i)==dt/dx(i)

```
Do i = n downto 1 step -1
    if (jump(i) == true) then
        α_x(i) += 1.0 *α_tval(tctr);
        α_tval(tctr-1) += 1.0 *α_tval(tctr);
        α_tval(tctr) = 0; tctr := tctr – 1;
```

# The Reverse Mode (RM) of AD – now for t = t+x(i)

TECHNISCHE UNIVERSITÄT DARMSTADT

Associate an "adjoint" $\alpha$ with every program variable and apply the following rules to the "inverted" program:
s = f(v,w) results in **$\alpha$_v += ds/dv* $\alpha$_s; $\alpha$_w += ds/dw* $\alpha$_s; $\alpha$_s := 0;**

```
t := 1.0;
Do i = 1 to n step 1
    If (t > 0) then
        jump(i):= true;
        t := t + x(i);
```

**1. Step: Make program reversible**

- Here by transformation into Single Assignment Form
- jump(1:n) initialized to 'false'

**2. Step: Adjoint computation**

- Initially $\alpha$_t :=1.0, $\alpha$_x(i) := 0 for i=1,…n.
- Upon exit, $\alpha$_x(i) == dt/dx(i)

```
Do i = n downto 1 step -1
    if (jump(i) == true) then
        α_x(i) += α_t;
```



SCIENTIFIC COMPUTING

# Remarks on AD modes (inputs x, outputs y)

TECHNISCHE UNIVERSITÄT DARMSTADT

## FM = Forward Mode (Tangent Linear Model)

- computes $dy/dx*S$ ($S$ is called "seed matrix") by propagating sensitivities of intermediate values with respect to input values.
- For $p$ input values of interest, runtime and memory scale at most with $p$.
- **Sparse/structured** Jacobians or gradients based on a sparse/structured problem are much cheaper to compute.

## RM = Reverse Mode (Adjoint Code)

- computes $W^T*dy/dx$ by propagating sensitivities of output values with respect to intermediate values.
- For $q$ output values of interest, RM runtime scales with $q$.
  Great for computing **„long gradients"**.
- Memory requirements are harder to predict, and greatly depend on structure of program. **Checkpointing** techniques keep memory usage under control.

# Building AD tools – for example ADiMat

TECHNISCHE UNIVERSITÄT DARMSTADT

- AD is an example of a rule-based, potentially context-sensitive, semantic augmentation of a computer program.

- Tool design issues:
  - Exploiting program structure and data flow information.
  - Managing the parallel world of derivatives.
  - Exploiting higher levels of abstraction in programming languages.

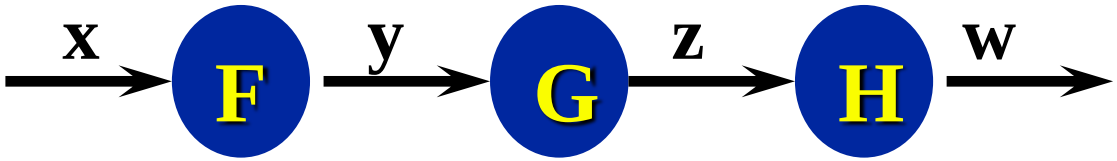
- ADiMat (AD for Matlab, Johannes Willkomm)
  - Particular challenges in particular for RM due to weak typing, polymorphism, automatic shape changes and higher-level operators.

# AD-enabled parallelism

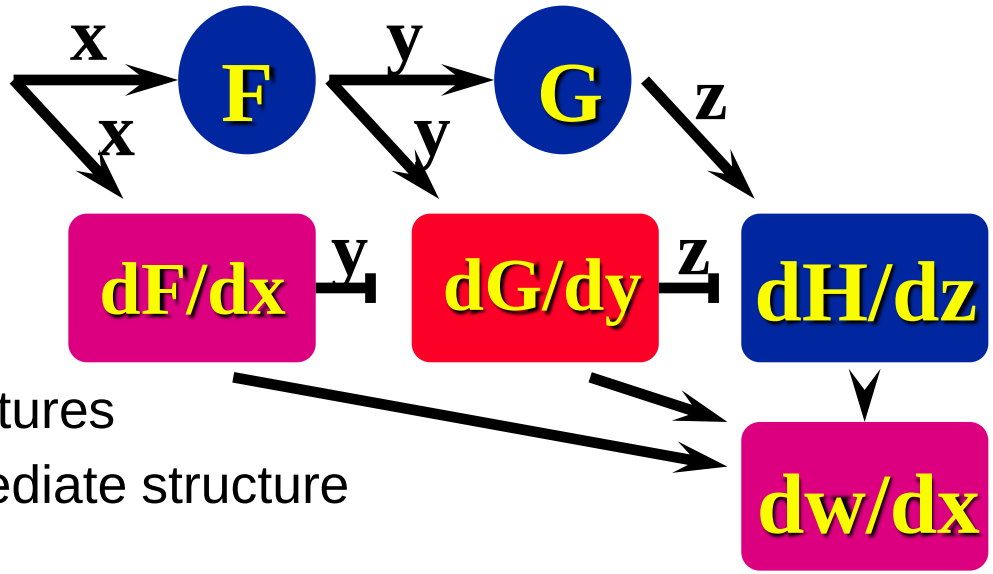
TECHNISCHE UNIVERSITÄT DARMSTADT

- The associativity of the chain rule can remove any serial dependence that exists in the the function code for the computation of derivatives.

**Function: inherently serial**




**Derivatives: in parallel!**




- Works at any granularity
- Applies to arbitrary computational structures
- Can expose intermediate structure



SCIENTIFIC COMPUTING

# Domain-specific languages

TECHNISCHE UNIVERSITÄT DARMSTADT

- Example **Dense Linear Algebra**
  - **Formal Linear Algebra Methods Environment**
    http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage
    Robert van de Geijn, CS Dept., UT Austin

- Example **Geometric Algebra (GA)**


Dietmar Hildenbrand, TU Darmstadt

- Example **Carbon Nanotube Structures**
  - Beginning collaboration of SC (Michael Burger) and Christian Schröppel/Jens Wackerfuß of MISMO Young Investigators Group at TU Darmstadt

# Formal Linear Algebra Methods Environment

TECHNISCHE UNIVERSITÄT DARMSTADT

| Step | Annotated Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$ |
|---|---|
| 1a | $\left\{ A = \hat{A} \right\}$ |
| 4 | Partition $A \rightarrow \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $L \rightarrow \left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right)$ where $A_{TL}$ and $L_{TL}$ are $0 \times 0$ |
| 2 | $\left\{ \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T \end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T \right\}$ |
| 3 | while do |
| 2,3 | $\left\{ \left( \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T \end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T \right) \wedge (m(A_{TL}) < m(A)) \right\}$ |
| 5a | Repartition $\left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)$ where $\alpha_{11}$ and $\lambda_{11}$ are scalars |
| 6 | $\left\{ \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c\|c\|c} L_{00} & \star & \star \\ \hline l_{10}^T & \hat{\alpha}_{11} - l_{10}^T l_{10} & \star \\ \hline L_{20} & \hat{a}_{21} - L_{20}l_{10} & \hat{A}_{22} - L_{20}L_{20}^T \end{array}\right) \wedge \hat{A}_{00} = L_{00}L_{00}^T \right\}$ |
| 8 | $\alpha_{11} := \sqrt{\alpha_{11}}$<br>$a_{21} := a_{21}/\alpha_{11}$<br>$A_{22} := A_{22} - a_{21}a_{21}^T$ |
| 5b | Continue with $\left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)$ |
| 7 | $\left\{ \begin{array}{l} \left(\begin{array}{c\|c\|c} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c\|c\|c} L_{00} & \star & \star \\ \hline l_{10}^T & \lambda_{11} & \\ \hline L_{20} & l_{21} & \hat{A}_{22} - L_{20}L_{20}^T - l_{21}l_{21}^T \end{array}\right) \\ \wedge \left(\begin{array}{c\|c} \hat{A}_{00} & \star \\ \hline \hat{a}_{21} & \hat{A}_{22} \end{array}\right) = \left(\begin{array}{c\|c} L_{00}L_{00}^T & \star \\ \hline \lambda_{11}l_{21} & l_{21}l_{21} + \lambda_{11}^2 \end{array}\right) \end{array} \right\}$ |
| 2 | $\left\{ \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T \end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T \right\}$ |
| | endwhile |
| 2,3 | $\left\{ \left( \left(\begin{array}{c\|c} A_{TL} & \star \\ \hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c\|c} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T \end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T \right) \wedge \neg(m(A_{TL}) < m(A)) \right\}$ |
| 1b | $\left\{ A = L \wedge \hat{A} = LL^T \right\}$ |

- http://z.cs.utexas.edu/wiki/flame.wiki/FrontPage
- The formal specification naturally embodies the „think in blocks“ paradigm that is required for memory locality and data reuse.
- **Invariants ensure correctness** (Hoare Logic).
- From this specification parallel code can automatically be generated for a variety of platforms.

# Developing performance models

TECHNISCHE UNIVERSITÄT DARMSTADT

- CATWALK Project in the DFG SPPEXA framework aims to “provide a flexible set of tools to support key activities of the performance modeling process”. http://www.sppexa.de/general-information/projects.html#CATWALK

- The goal is to make performance modeling accessible
  - During initial code development
  - During software lifecycle management

  to identify potential scaling problems.

- SC (Christian Iwainsky, Artur Mariano) is involved in extending performance models to manycore architectures with NUMA performance characteristics.

# InstRO (Instrumentation with Rose)
# A Toolkit for Program Instrumentation Tools

TECHNISCHE UNIVERSITÄT DARMSTADT

- Different performance analysis tools require different data, typically through tracing or sampling.

- Current instrumentation techniques do not take the structure of programs into account – a real problem for large-scale parallel codes!

- InstRO (SC, Christian Iwainsky)
  - Harness the power of compiler-based code analysis for scalable performance analysis tools.
  - Provide tool developers with a framework to rapidly develop customized instrumentation tools.
  - Separate selection and code modification phases to prevent selection of new code generated by the tool and to enable tool specific backends.

# SS12: Praktikum „Automatic code generation and programming“

TECHNISCHE UNIVERSITÄT DARMSTADT

- In this practical session the design of language extensions and their implementation will be explored.
- The fundamental concepts of
  - program augmentation (exemplified by automatic differentiation) and
  - parallelization (exemplified by directive-based language extensions like OpenMP or OpenACC) will be approached on the basis of a simple subset language of C or Matlab.

# CONCLUSIONS

# Conclusions

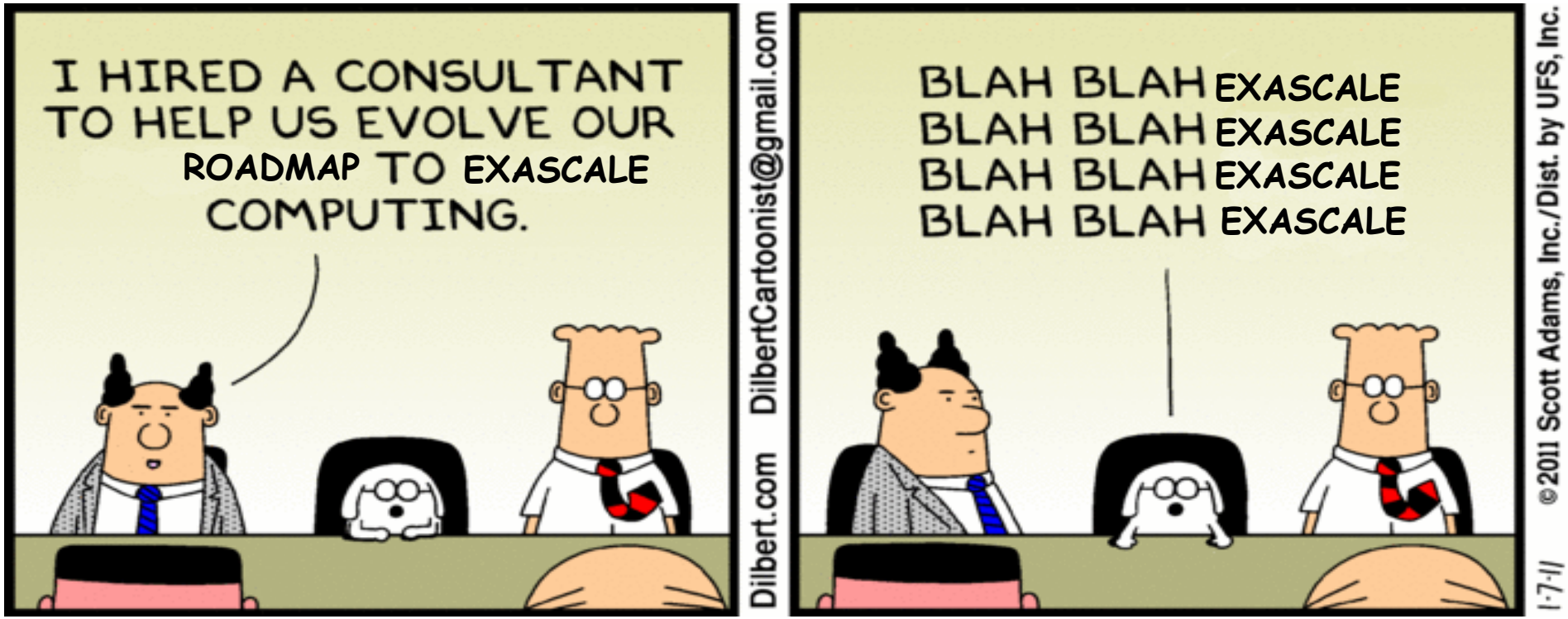
TECHNISCHE UNIVERSITÄT DARMSTADT

- **Software is the key for innovation and economics in HPC.**
  - Moore‘s law will continue to provide us with a great variety of interesting architectures.
  - Exploiting the resulting freedom of choice, software becomes ever more important in the algorithms-software-hardware codesign triad.

- **Sensible use of ever more powerful HPC resources requires:**
  - Algorithmic innovations in modeling and numerics.
  - New paradigms for software development.
  - New paradigms for HPC software stewardship

# Paraphrasing Dilbert

(modified cartoon courtesy of Thomas Lippert, FZ Jülich)




"Blah Blah Exascale!" unless we close the software gap