Das Software-Loch im Hochleistungsrechnen

Christian Bischof
FG Scientific Computing
Hochschulrechenzentrum
TU Darmstadt

SCIENTIFIC COMPUTING

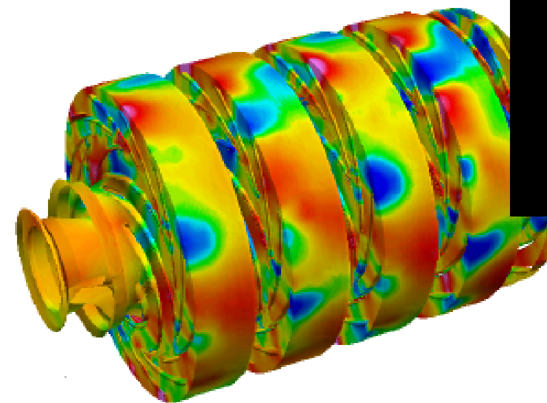
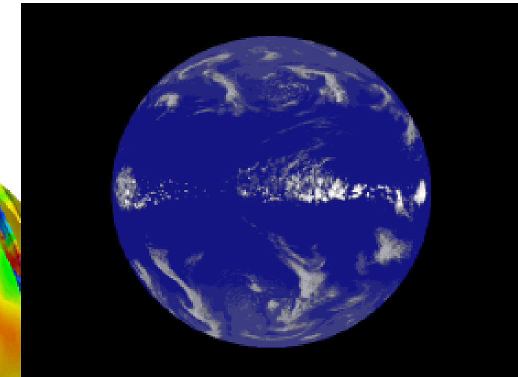
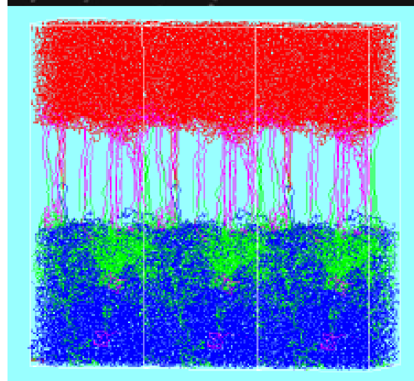
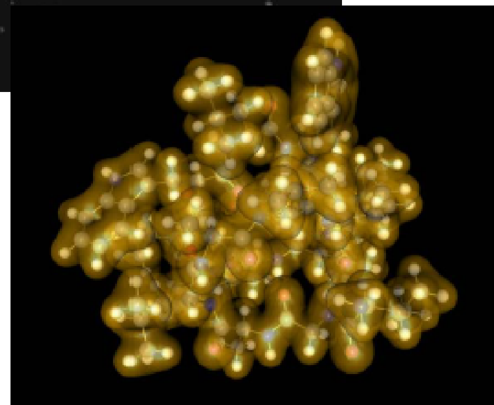
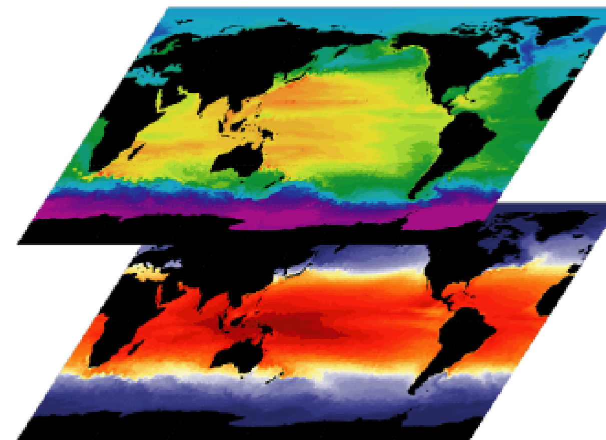
WORUM GEHT ES BEIM HOCHLEISTUNGSRECHNEN?

Hochleistungsrechnen

- FAZ am 3. Juni 2012: „Die Sehnsucht der Forscher nach den Exaflops“ (= 10^18 floating point operations per second).
  - „Die Rechenkraft von Supercomputern steigt und steigt. Und die Wissenschaft zählt darauf, dass das auch in Zukunft so weitergeht.“

- Wie kommt man in der Forschung auf solche Zahlendimensionen?
  - Zum Vergleich: Der menschliche Körper enthält ca. 10^13 Zellen, also ein Millionstel von 10^18 (http://en.wikipedia.org/wiki/Human_microbiome).
  - Oder eine Million Menschen enthalten so viele Zellen wie ein Exaflop-Computer in einer Sekunde Operationen ausführt.

Beispiele für „Grand Challenges“ im wissenschaftlichen Rechnen

Graduiertenschule CE und Exzellenzcluster CSI

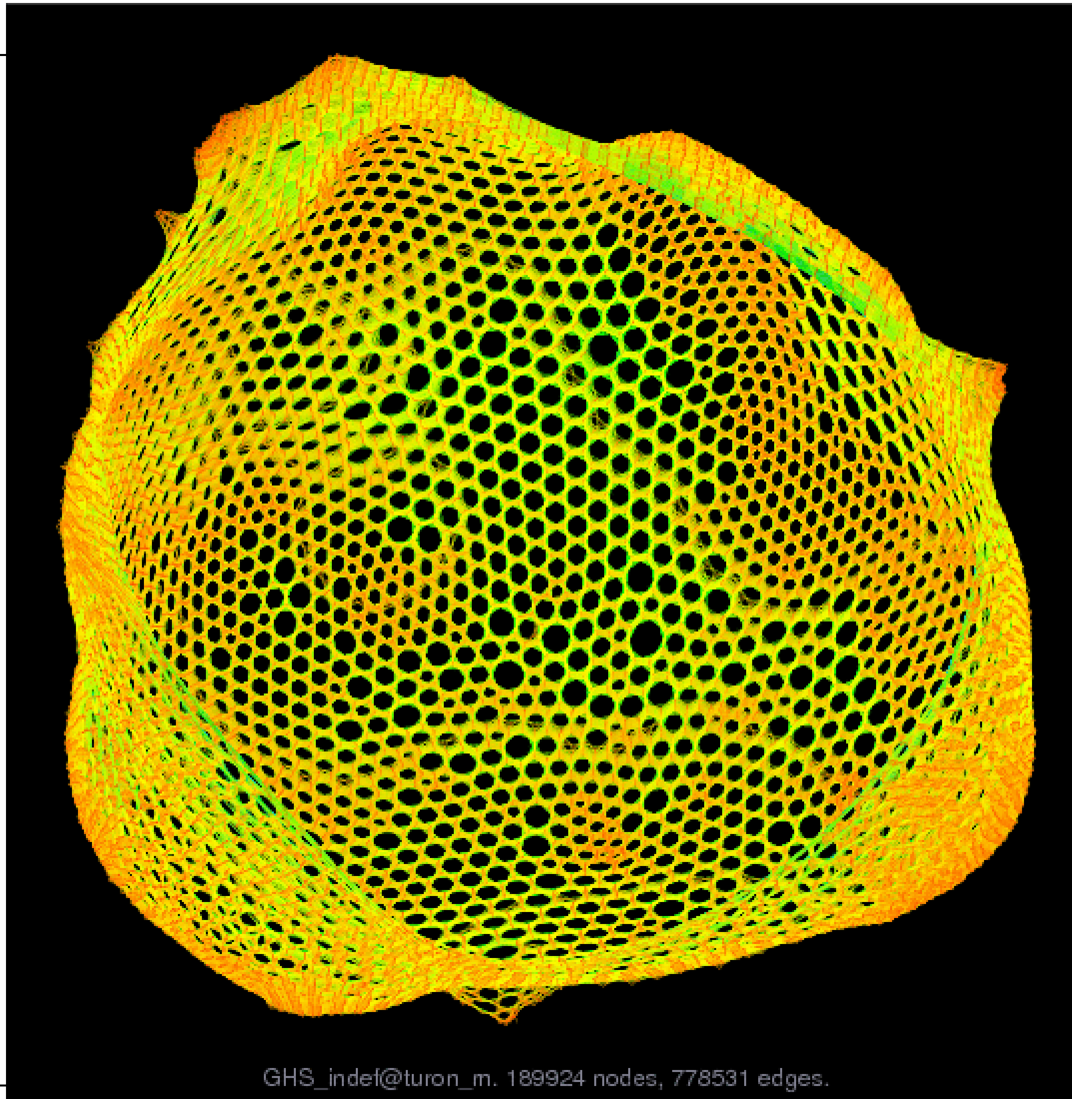
- CE = Computational Engineering, http://www.graduate-school-ce.de/
  - Graduiertenschule in der Exzellenzinitiative, gerade verlängert.
  - Thematische Fokussierung auf Simulationswissenschaft im Kontext der Ingenieurwissenschaften.
  - Siehe auch „forschen“, Nr. 2/2011, http://www.tu-darmstadt.de/vorbeischauen/publikationen/forschung/archiv/aktuellethemaforschung_3136.de.jsp
- CSI = Cluster Smart Interfaces, http://www.csi.tu-darmstadt.de/
  - Cluster in der Exzellenzinitiative.
  - Thematische Fokussierung auf Verständnis und Design von Flüssigkeitsgrenzen.
  - Siehe auch „forschen“, Nr. 2/2009, http://www.tu-darmstadt.de/vorbeischauen/publikationen/forschung/archiv/aktuellethemaforschung_1344.de.jsp
- **Wichtige Treiber für das Hochleistungsrechnen (high performance computing, HPC) an der TU Darmstadt.**

Das Vorgehen in „Computational Science und Engineering“ (Rechnergestützte Wissenschaften, Simulation Science)

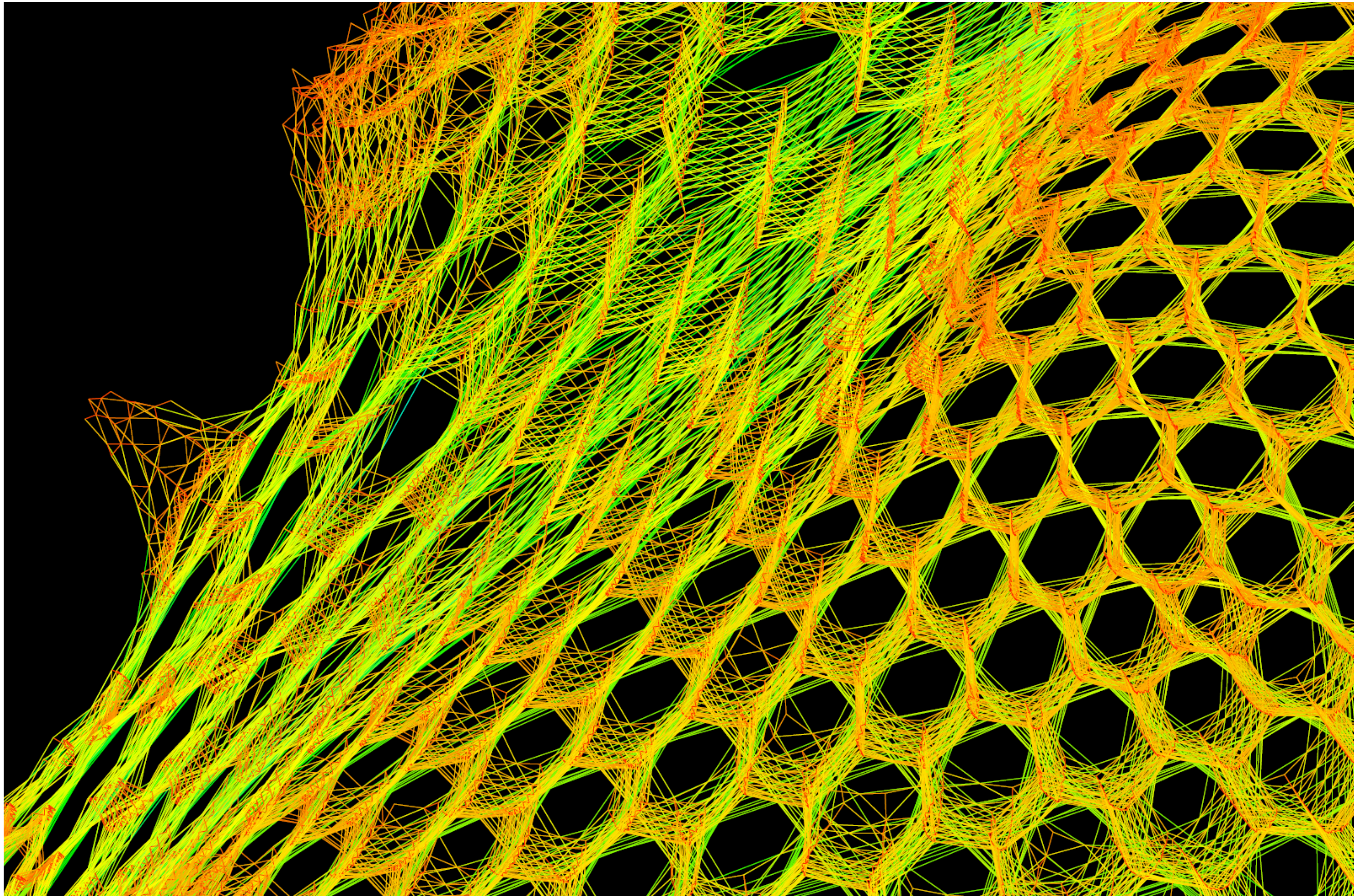
1. Modellierung des Problems, z.B. durch
   - Partielle Differentialgleichungen
   - Zelluläre Automaten
   - Teilchenmengen
2. Umsetzung in geeignete Rechenvorschriften (Algorithmen)
   - Eine ganz wichtige Rolle spielt hier die Lösung linearer Gleichungssysteme $A * x = b$
   - Die Matrizen können sehr groß werden, z.B. 100 Milliarden Einträge.
3. Entwicklung von Software für einen Rechner
   - Datenstrukturen
   - Algorithmen

SC SCIENTIFIC COMPUTING

Ein Beispiel für eine „Diskretisierung“



- Aus „Florida Test Matrix Collection“, http://www.cise.ufl.edu/research/sparse/matrices/
- GHS_indef/turon_m
- Hintergrund ist Modellierung des Untergrundes einer Mine
- Berechnung von Flussphänomenen in porösen Medien
- 1,7 Mio Nichtnullen

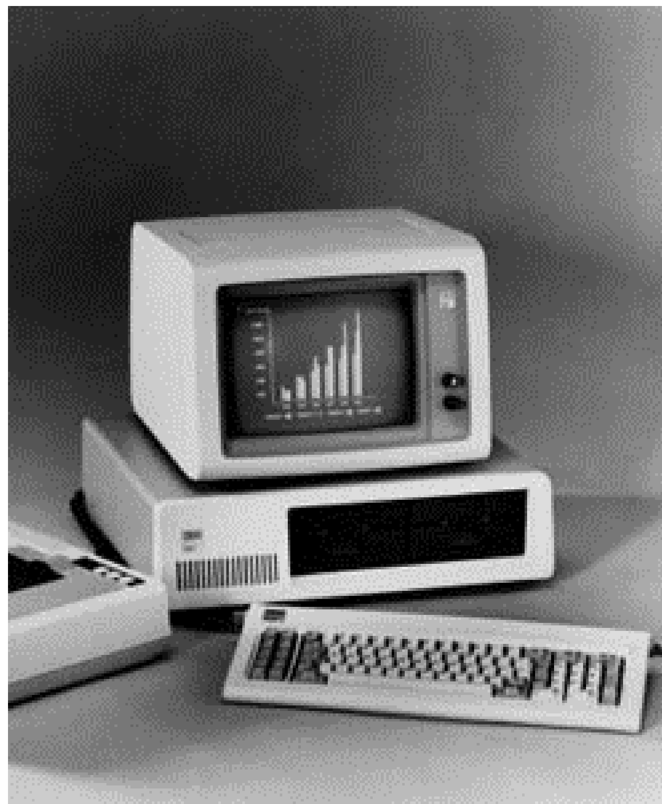


WIE SCHNELL KANN MAN EIGENTLICH RECHNEN

Ein Schritt zurück: Der IBM PC (1981)

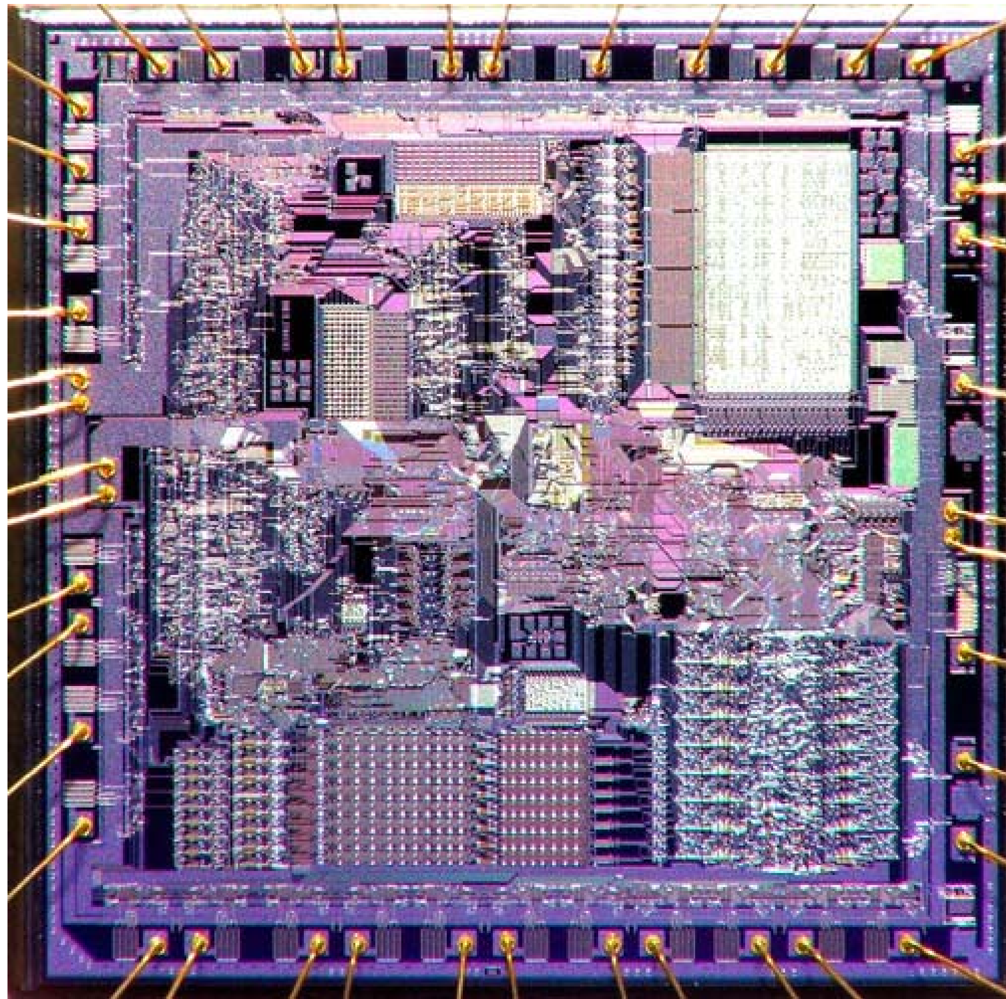
TECHNISCHE UNIVERSITÄT DARMSTADT

PC = **Personal** Computer



Vorher war der Rechner etwas besonderes und weit weg, von Operateuren betreut.

In dem IBM PC steckte der Intel 8088 Chip



- 29000 Transistoren
- Taktrate 4,77 Mhz
- Maximal 256 kByte RAM

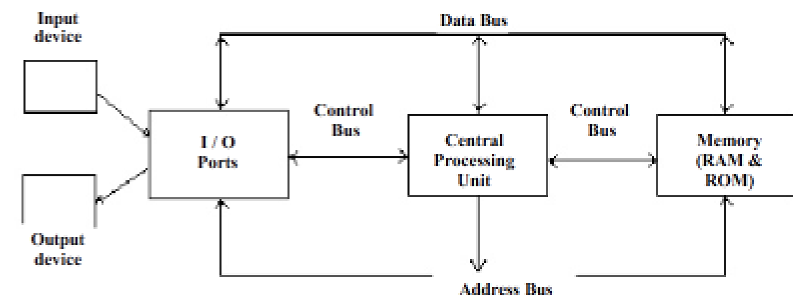


Fig. 1 Block Diagram of a simple Computer or a Microcomputer.

Der erste Supercomputer: Cray 1 (1976)

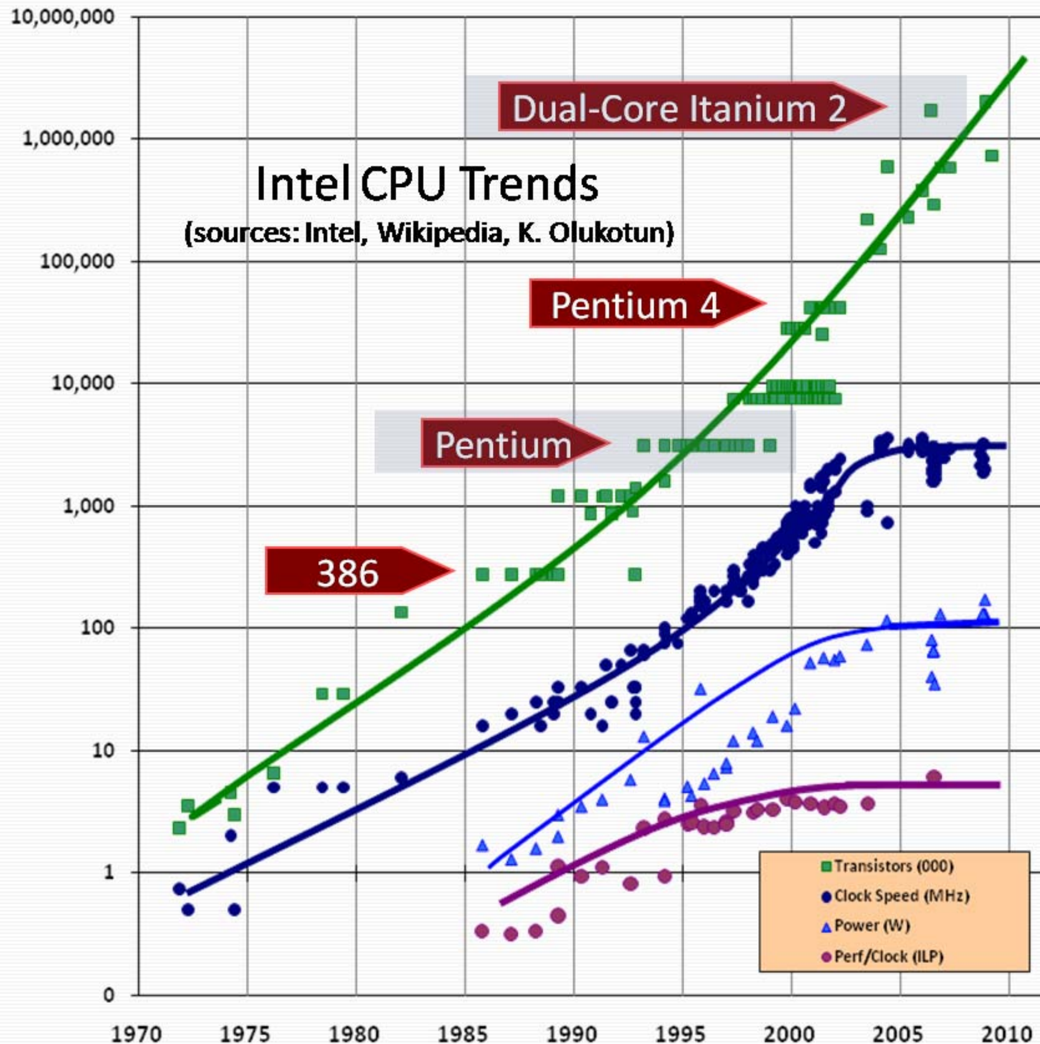
TECHNISCHE UNIVERSITÄT DARMSTADT

- Cray-1 im Deutschen Museum
- Erste Maschine in Los Alamos (zur Atombomben-Forschung)
- 5,5 Tonnen
- 80 MHz Taktrate
- 8 Mbyte RAM
- 8,8 Mio $

- Ein Spezialrechner für die Wissenschaft
- Im „Keller“: Kaltwasser, Strom

SC SCIENTIFIC COMPUTING

Das Moore'sche Gesetz



Computer werden alle 18 Monate doppelt so schnell

Sowohl die personal computer als auch daraus abgeleitete Server

Der Grund:

- **Anzahl der Transistoren verdoppelt sich alle 2 Jahre**
- **Die Taktrate steigt**

… aber jetzt steigt die Taktrate nicht mehr!

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

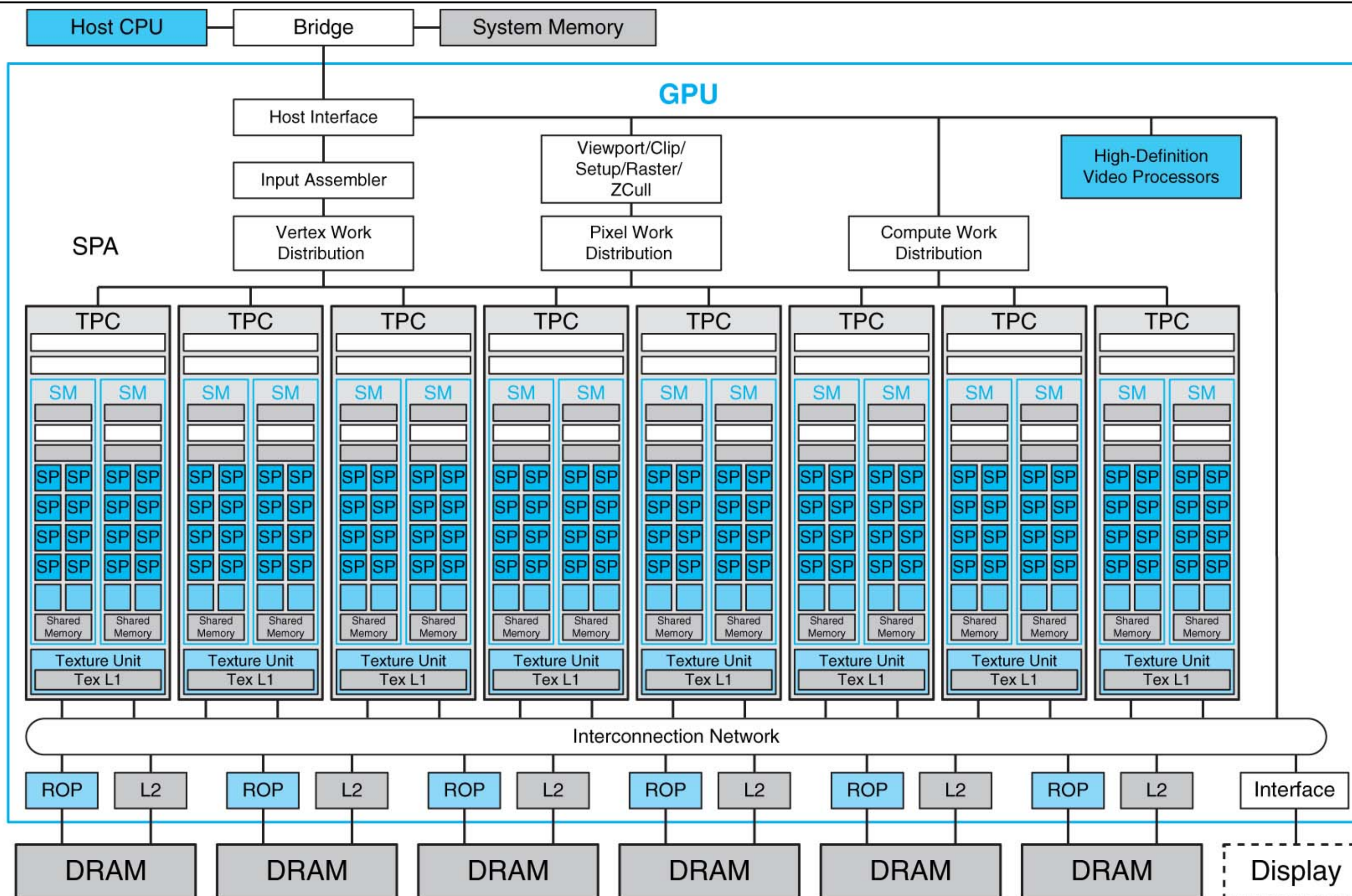
TECHNISCHE UNIVERSITÄT DARMSTADT

SCIENTIFIC COMPUTING

Die Welt des HPC sieht prima aus!

- Moore'sches Gesetz: Die Anzahl der Transistoren für die gleiche Investitionssumme verdoppelt sich alle 18 Monate.
  - Leider verdoppelt sich die Speicherbandbreite nur alle 6 Jahre.
- Prozessoren gibt es in vielen „Geschmacksrichtungen“:
  - **Alle beinhalten mehrere Rechenkerne (cores)**
  - GPGPUs, wie NVIDIA GeForce 8800
  - Many-core, wie AMD Barcelona

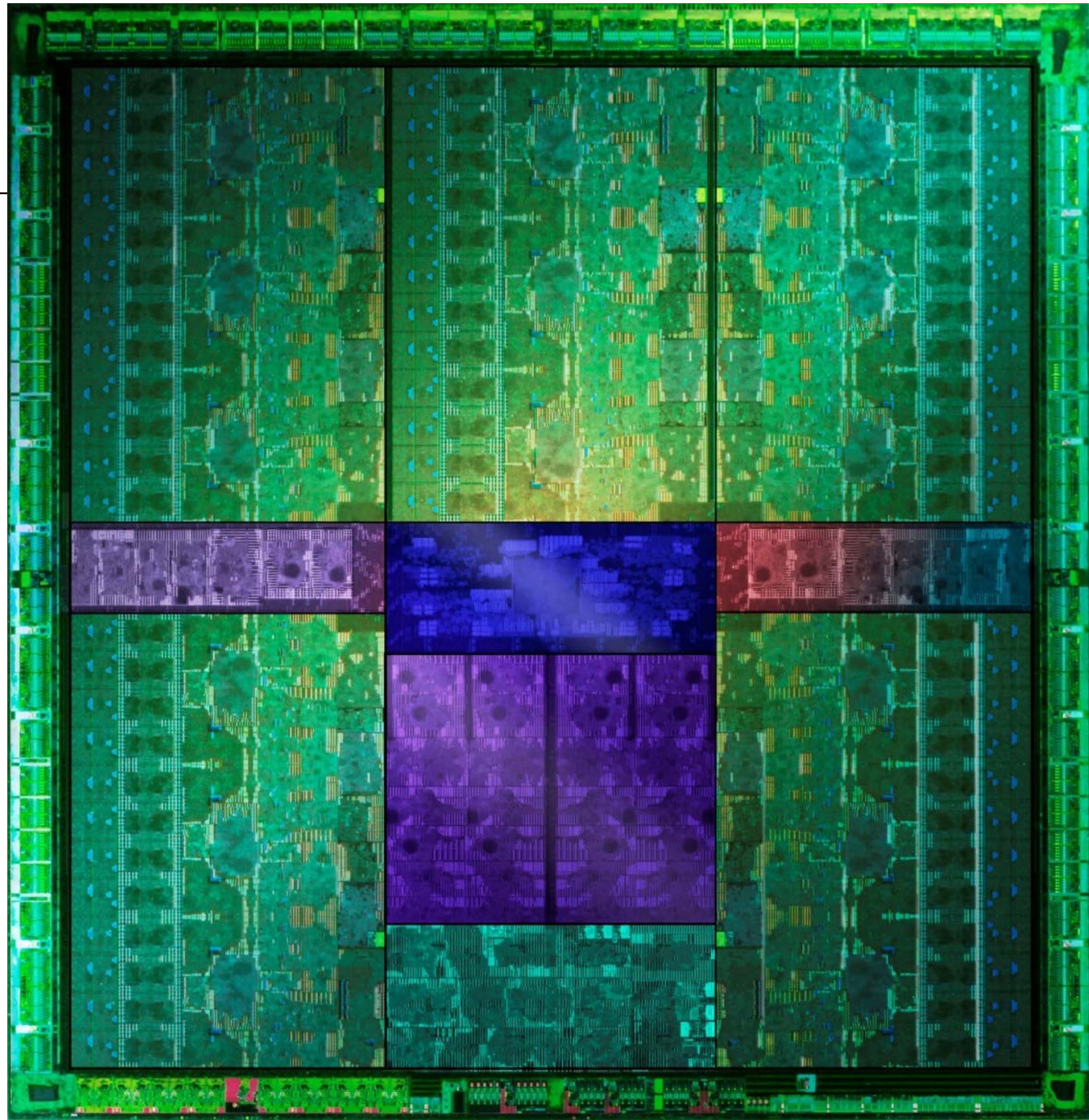
NVIDIA GeForce 8800



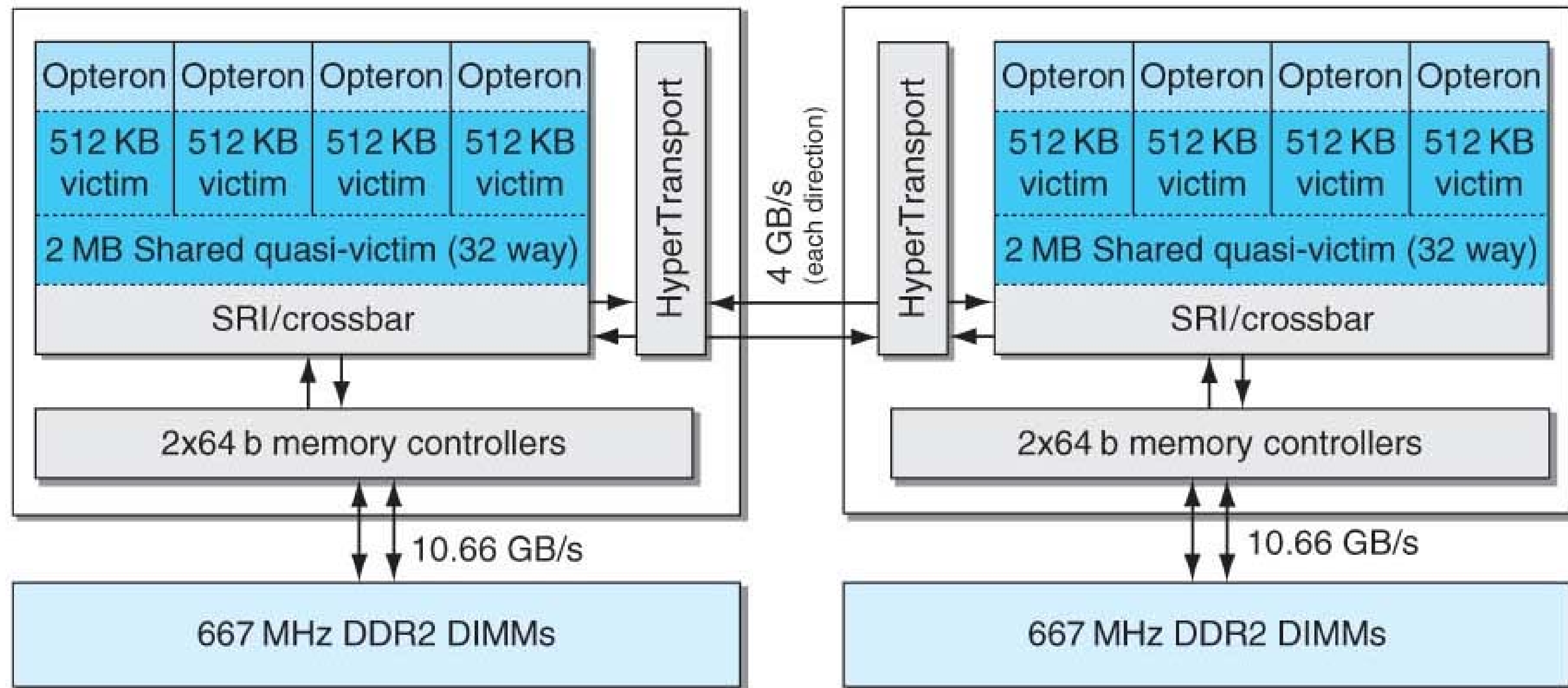
Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann, 2011

NVIDIA Kepler Chip Layout

- 7,1 Milliarden Transistoren
- getaktet mit 1,3 GHz
- 1536 Rechenkerne



AMD Opteron X4 2356 (Barcelona)



Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann, 2011

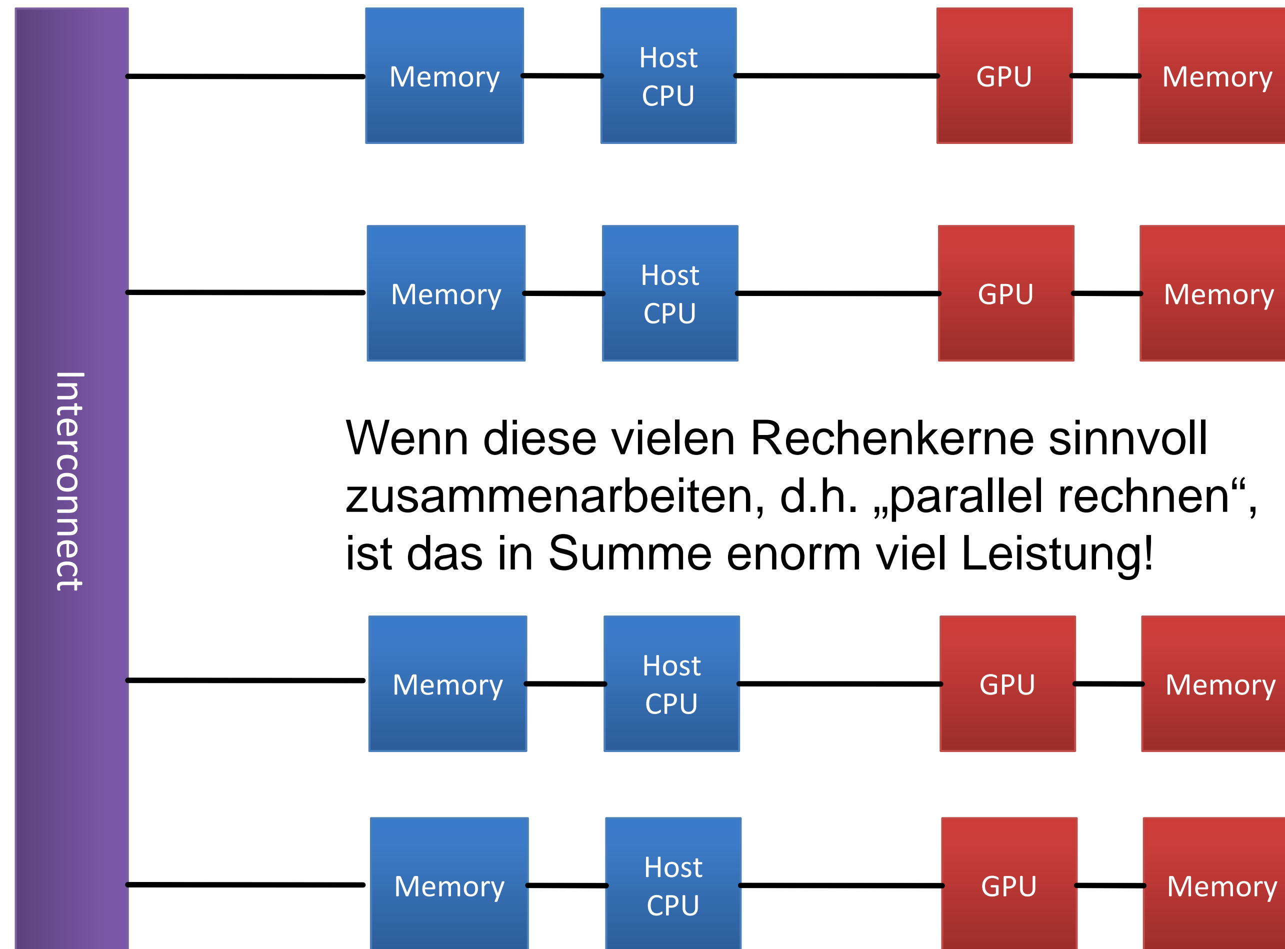
Randbedingungen für Chip-Design

TECHNISCHE UNIVERSITÄT DARMSTADT

- Stromverbrauch (und Abwärme!) skaliert linear mit der Anzahl der Transistoren, aber quadratisch mit der Taktrate.
  - Doppelt so viel Transistoren verbrauchen doppelt so viel Strom.
  - Ein Chip mit der halben Taktrate verbraucht nur ein Viertel so viel Strom.
- **Schlussfolgerung:**
  - **Taktrate senken!**
  - **Mehrere Rechenkerne (=cores) auf einen Chip packen!**
- Beispiele:
  - Intel Xeon Westmere-EX hat 10 cores und 2,6 Milliarden Transistoren (ca. 100.000 mal soviel wie Intel 8086), getaktet mit 2,4 GHz.
  - NVIDIA Kepler hat 1.536 cores und 7,1 Milliarden Transistoren, getaktet mit 1,3 GHz.

HPC aus Standardbauteilen

TECHNISCHE UNIVERSITÄT DARMSTADT



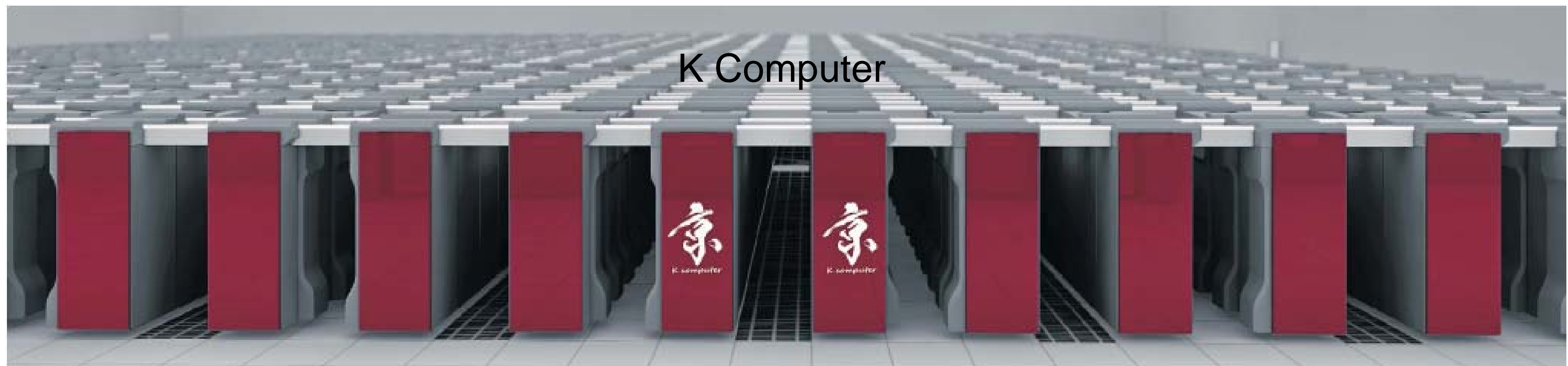
Wenn diese vielen Rechenkerne sinnvoll zusammenarbeiten, d.h. „parallel rechnen“, ist das in Summe enorm viel Leistung!

SC SCIENTIFIC COMPUTING

Die schnellsten Rechner

	NAME	SPECS	SITE	COUNTRY	CORES	R_{max} Pflop/s
1	**Sequoia**	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	DOE / NNSA / LLNL	USA	1,572,864	16.33
2	**K computer**	Fujitsu SPARC64 VIIIfx 2.0GHz, Tofu interconnect	RIKEN AICS	Japan	705,024	10.51
3	**Mira**	IBM BlueGene/Q, Power BQC 16C 1.60 GHz, Custom interconnect	DOE / SC / ANL	USA	786,432	8.153
4	**SuperMUC**	IBM iDataPlex DX360M4, Xeon E5-2680 8C 2.70GHz, Infiniband QDR	Leibniz Rechenzentrum	Germany	147,456	2.897
5	**Tianhe-1A**	NUDT YH MPP, Xeon X5670 6C 2.93 GHz, NVIDIA 2050	NUDT/NSCC/Tianjin	China	186,368	2.566

Wie wird Rechenleistung (Rmax) gemessen?



SC SCIENTIFIC COMPUTING

Measuring top speed

Computers are 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

Das Gauß-Verfahren, aus Bronstein und Semendjajew: Taschenbuch der Mathematik, 1957 (1979 in der 19. Ausgabe erschienen)

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 bezeichnen 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

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

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

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

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

Eine optimierte Implementierung der Linpack-Benchmark umfasst 75,000 Zeilen Quellcode!
(Ruud van der Pas,
Oracle, pers. Kommunikation)

Was die Codes so kompliziert macht, ist die Organisation der Datenzugriffe.

Gute und schlechte Nachrichten

- Gute Nachrichten: In Anbetracht der Komplexität und Größe der Rechensysteme ist es erstaunlich, dass man die Linpack-Benchmark so schnell lösen kann.
- Schlechte Nachrichten: Es ist wissenschaftlich irrelevant!
  - Ich kenne keine Anwendung, bei der man so große Gleichungssysteme mit dem Gauß-Algorithmus löst.
  - O(n*n*n) Gleitpunktoperationen mit O(n*n) Daten können auch mit langsamen Speichern schnell laufen.
  - Bei einer schnellen Fouriertransformation mit O(n log n) Gleitpunktoperationen auf O(n) Daten sieht die Welt ganz anders aus!
- Herausforderung: **Parallele Programmierung von Hochleistungs-rechnern für wissenschaftlich relevante Anwendungen!**

SCIENTIFIC COMPUTING

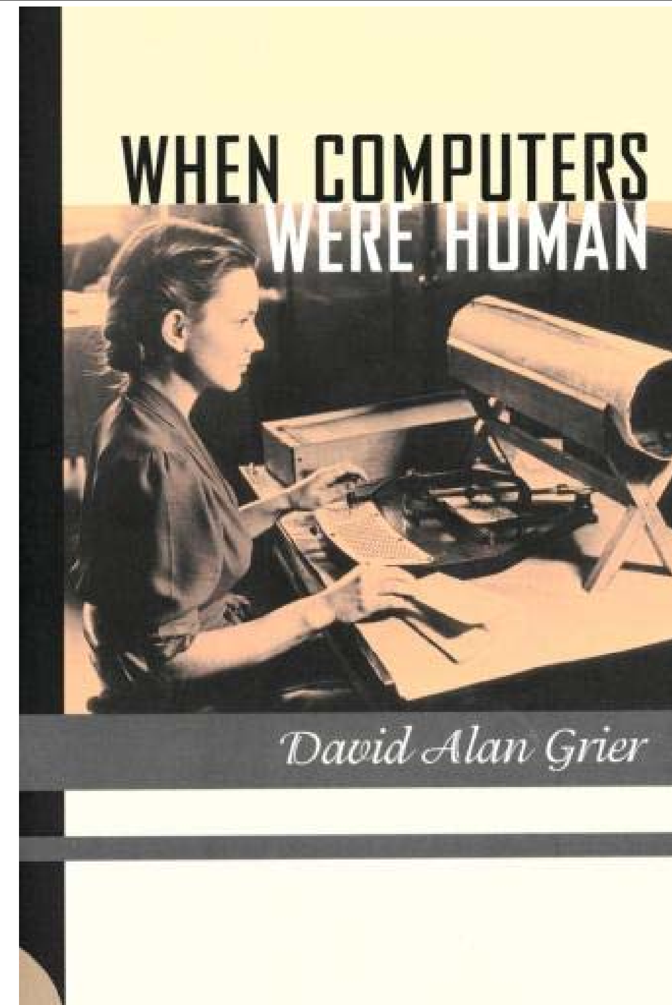
PARALLELES RECHNEN IST KEINE NEUE IDEE

Der Beruf des Computer

When Computers
were human

David Alan Grier, 2005

Die Geschichte eines
Berufsstandes



Dank an Thomas Ludwig, DKRZ

Der Beruf des Computers…

- Der Beruf existierte vom Beginn des 19. Jahrhunderts bis zum Ende des Zweiten Weltkrieges

- Hauptsächliche Berechnungsgegenstände
  - Bewegungsbahnen für Himmelskörper (Ephemeriden)
  - Nautische Handbücher für die Seefahrt
  - Tabellen für Artilleriegeschosse

- Parallele Human-Computer
  - Computing offices / Computing laboratories
  - Fassen 5-150 menschliche Computer zusammen

Dank an Thomas Ludwig, DKRZ

Menschliche Flops

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wieviele Flops leistet der Mensch?
- Grob geschätzt: 1 Flops pro 1,5 Minuten
  also 1/100 Flops
- Wenn Ihnen das als zu wenig vorkommt:
  Berechnen Sie 0,283765 x 0,847102 auf Zeit

Dank an Thomas Ludwig, DKRZ

SC SCIENTIFIC COMPUTING

Lewis Fry Richardson (1881-1953)

TECHNISCHE UNIVERSITÄT DARMSTADT

- Wetteraufzeichnungen seit 1870
- 1916 schreibt Richardson sein Buch *Weather Prediction by Arithmetical Finite Differences.*
- **Seine Vision:** *Perhaps some day in the dim future it will be possible to advance computations faster than the weather advances and at a cost less than the saving to mankind due to the information gained.*

Dank an Thomas Ludwig, DKRZ

Richardsons *Forecast Factory*

- Er entwickelt Differentialgleichungen für Temperatur, Feuchtigkeit, Druck usw.
- Teilt den Globus in 2000 Felder auf, an denen diese Eigenschaften alle 3 Stunden berechnet werden sollen.
- Er schätzt, dass er 32 Computer pro Feld benötigt, um die Zeitvorgabe einhalten zu können.
- Insgesamt also 64.000 Computer.

Dank an Thomas Ludwig, DKRZ

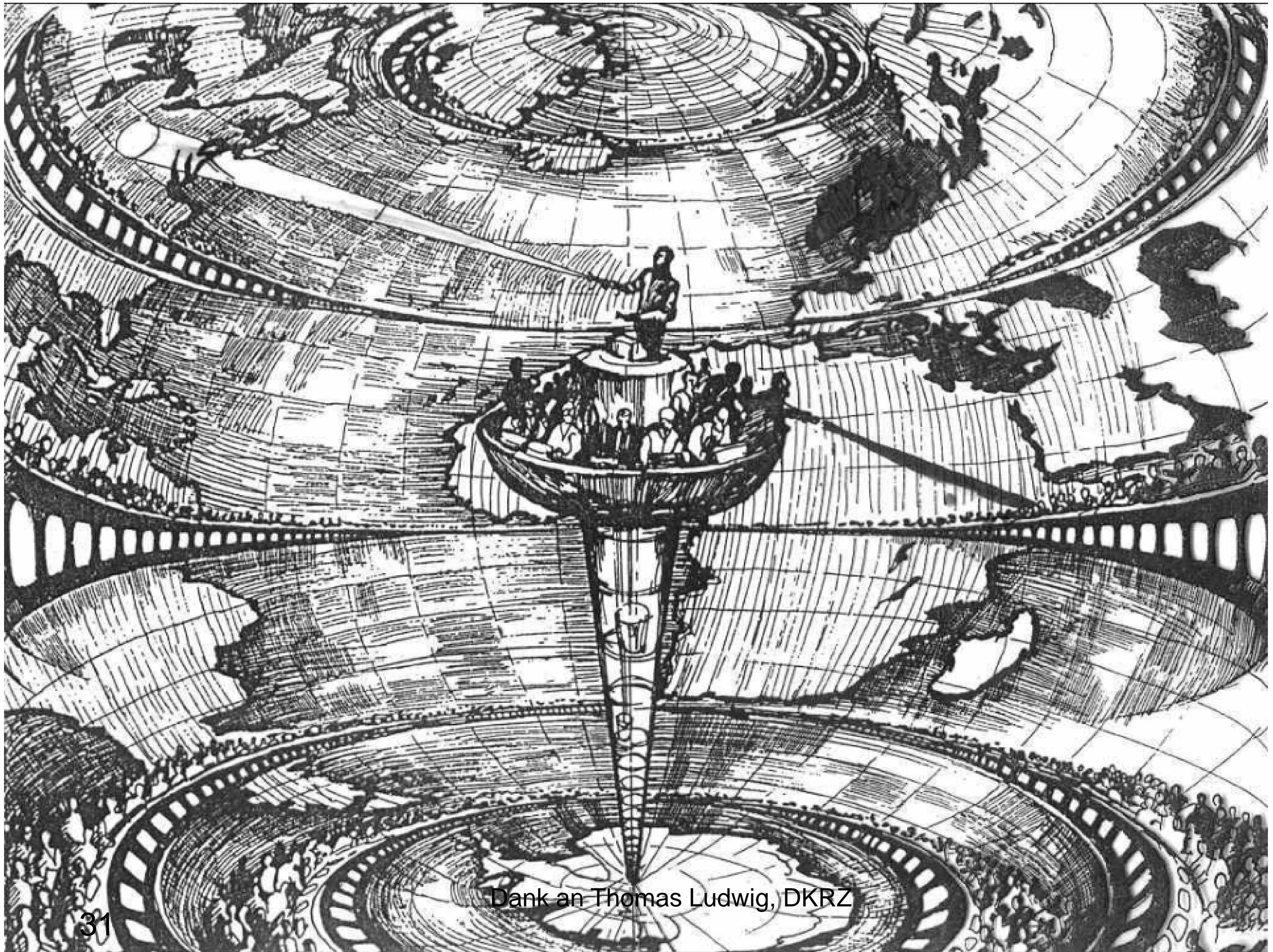
SCIENTIFIC COMPUTING

Richardsons *Forecast Factory*

TECHNISCHE UNIVERSITÄT DARMSTADT

- Ein gigantischer kugelförmiger Rechnerraum nimmt alle 64.000 Computer auf.
- Der Rechnerraum ist innen mit der Landkarte des Globus bemalt.
- Die Computer arbeiten auf Balkonen nahe den ihnen zugeordneten Punkten der Wetterberechnung.
- Sie signalisieren ihre Ergebnisse mit Lichtsignalen, sodass Dritte sie sehen können.
- In der Mitte steht eine große Säule, darauf ein erfahrener Computer.
- Er garantiert das gleichförmige Voranschreiten der Berechnung.
- Sendet rosa Lichtsignale zu Computern, die zu weit voraus sind, blaue Lichtsignale zu Computern, die zurückliegen.

Dank an Thomas Ludwig, DKRZ


Dank an Thomas Ludwig, DKRZ

TECHNISCHE UNIVERSITÄT DARMSTADT

PARALLELES PROGRAMMIEREN

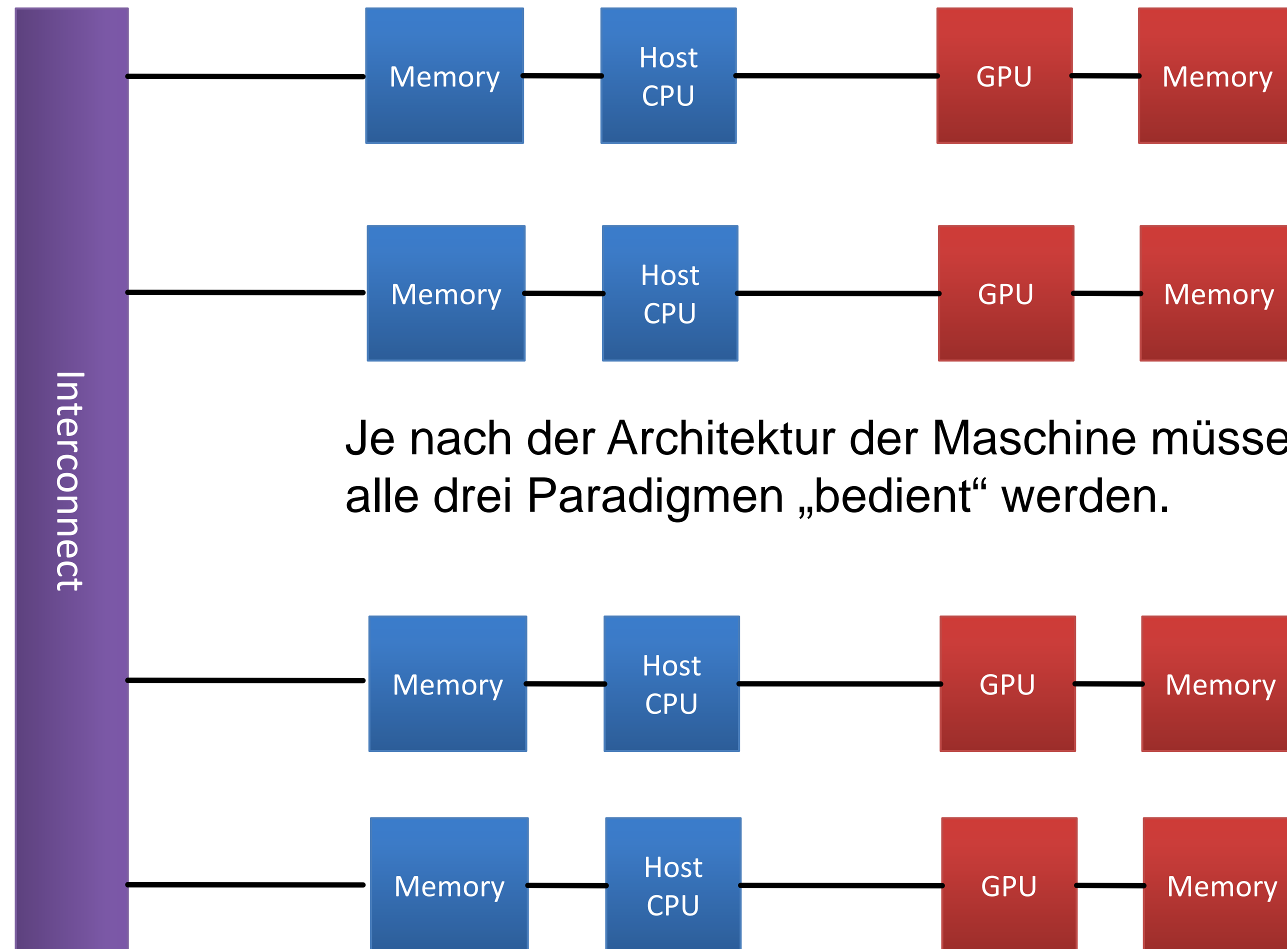
SC SCIENTIFIC COMPUTING

Software für Hochleistungsrechner

- Die Idee des parallelen Rechnens, um hohe Rechenleistung zu erzielen, ist nicht neu!
- Herausforderung: **„Paralleles Programmieren“, d.h. sinnvolle Orchestrierung der vielen funktionalen Einheiten („cores“), die möglichst unabhängig voneinander arbeiten sollen, aber am Ende ein sinnvolles Ergebnis für das betrachtete Problem liefern sollen**.
  - Richardson: sog. Master-Slave Paradigma

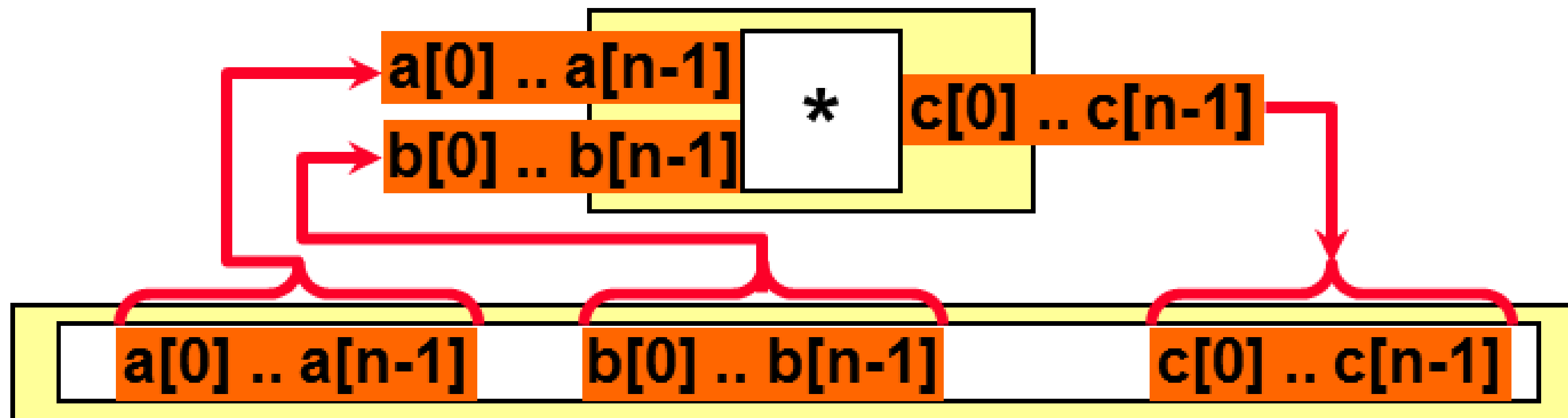
- **3 Architekturparadigmen:**
  - Vektorparallelität
  - Multithreading in einem gemeinsamen Speicher (shared memory)
  - Paralleles Rechnen mit einem verteilten Speicher

Paradigmen des parallelen Programmierens



Je nach der Architektur der Maschine müssen alle drei Paradigmen „bedient“ werden.

Parallelität: Vektor



Einfache Programmschleife:

```
double a[n], b[n],  c[n];
            :
for (i=0; i<n; i++)
  c[i] = b[i] * a[i];
```

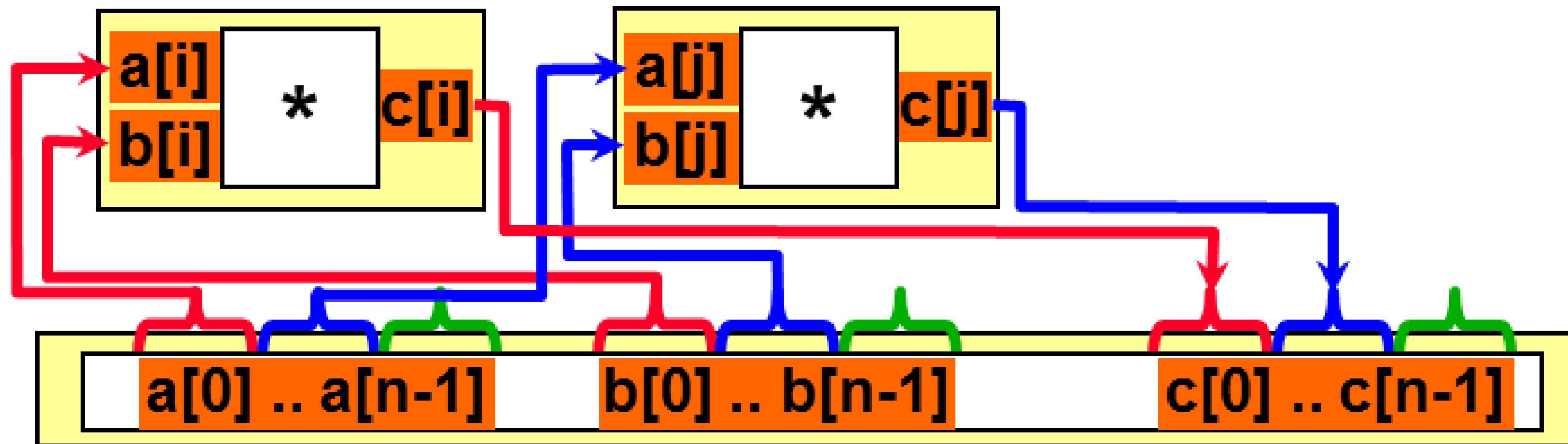
Ausführungseinheit: Vektoroperation

Die gleiche Operation wird auf verschiedenen Einträgen eines Vektors gleichzeitig durchgeführt.

Große Vektoren werden „gestückelt".

Parallelität: Shared-memory multithreading

TECHNISCHE UNIVERSITÄT DARMSTADT



Einfache Programmschleife:

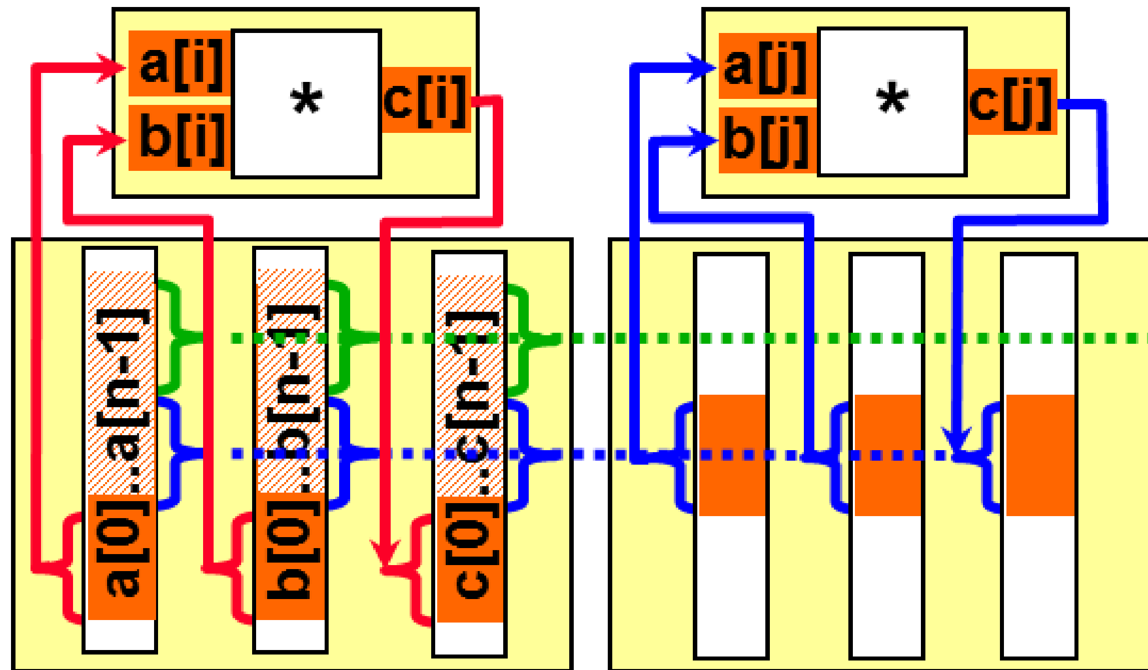
```
double a[n], b[n],  c[n];
           :
for (i=0; i<n; i++)
  c[i] = b[i] * a[i];
```

Ausführungseinheit: **Thread – leichtgewichtiger Prozess, der Daten verarbeitet.**

Verschiedene Threads greifen auf Daten in einem gemeinsamen Speicher zu.

Sind Threads unterschiedlich schnell, können die schnelleren Threads mehr Arbeit übernehmen.

Parallelität: Distributed-memory



Die Daten sind auf unterschiedliche Prozessoren und ihre Speicher verteilt.

Ein Prozessor sieht nur „seine“ Daten.

Datenaustausch zwischen verschiedenen Prozessoren muss vom Programmierer gemanagt werden.

```
double a[n], b[n],  c[n];
for (i=0; i<n; i++)
  c[i] = b[i] * a[i];
```

TECHNISCHE UNIVERSITÄT DARMSTADT

SHARED MEMORY PROGRAMMIERUNG MIT OPENMP

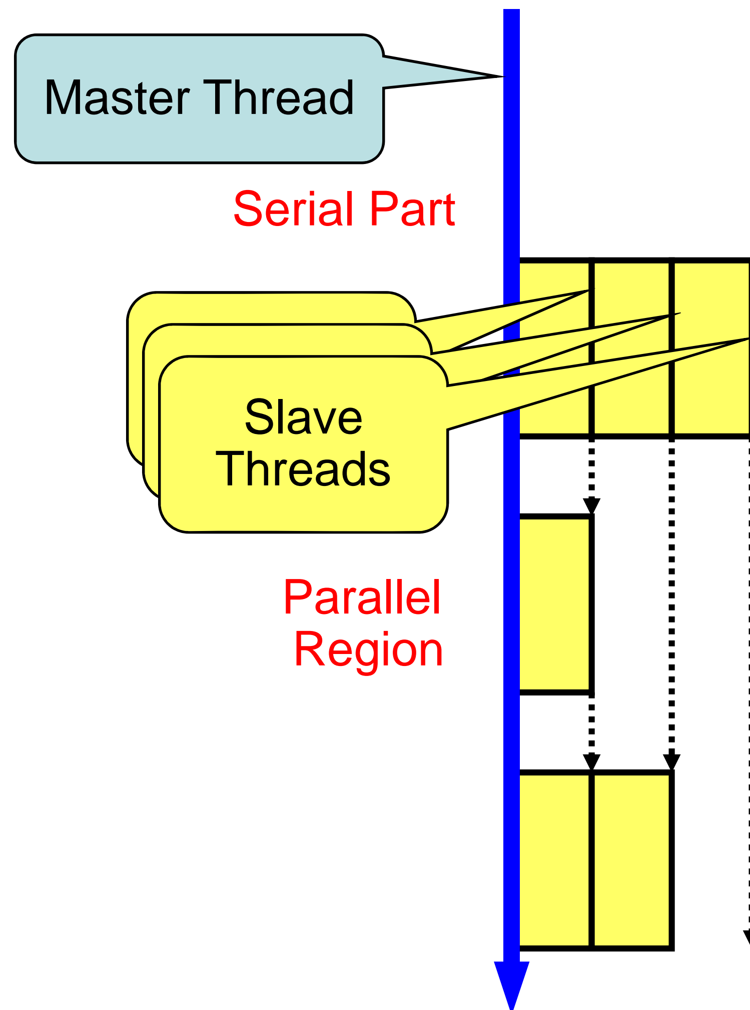
SC SCIENTIFIC COMPUTING

Programmierung von Shared Memory Multiprozessoren (SMP)

- Relativ komfortable Architekturabstraktion.
- Alle threads können auf einen gemeinsamen Speicher zugreifen.
- Das erleichtert die Arbeitsteilung, denn man kann die Arbeit dynamisch an die verschiedenen threads verteilen.
- De facto Programmierstandard: OpenMP (www.openmp.org)

The OpenMP Fork-Join Model

TECHNISCHE UNIVERSITÄT DARMSTADT



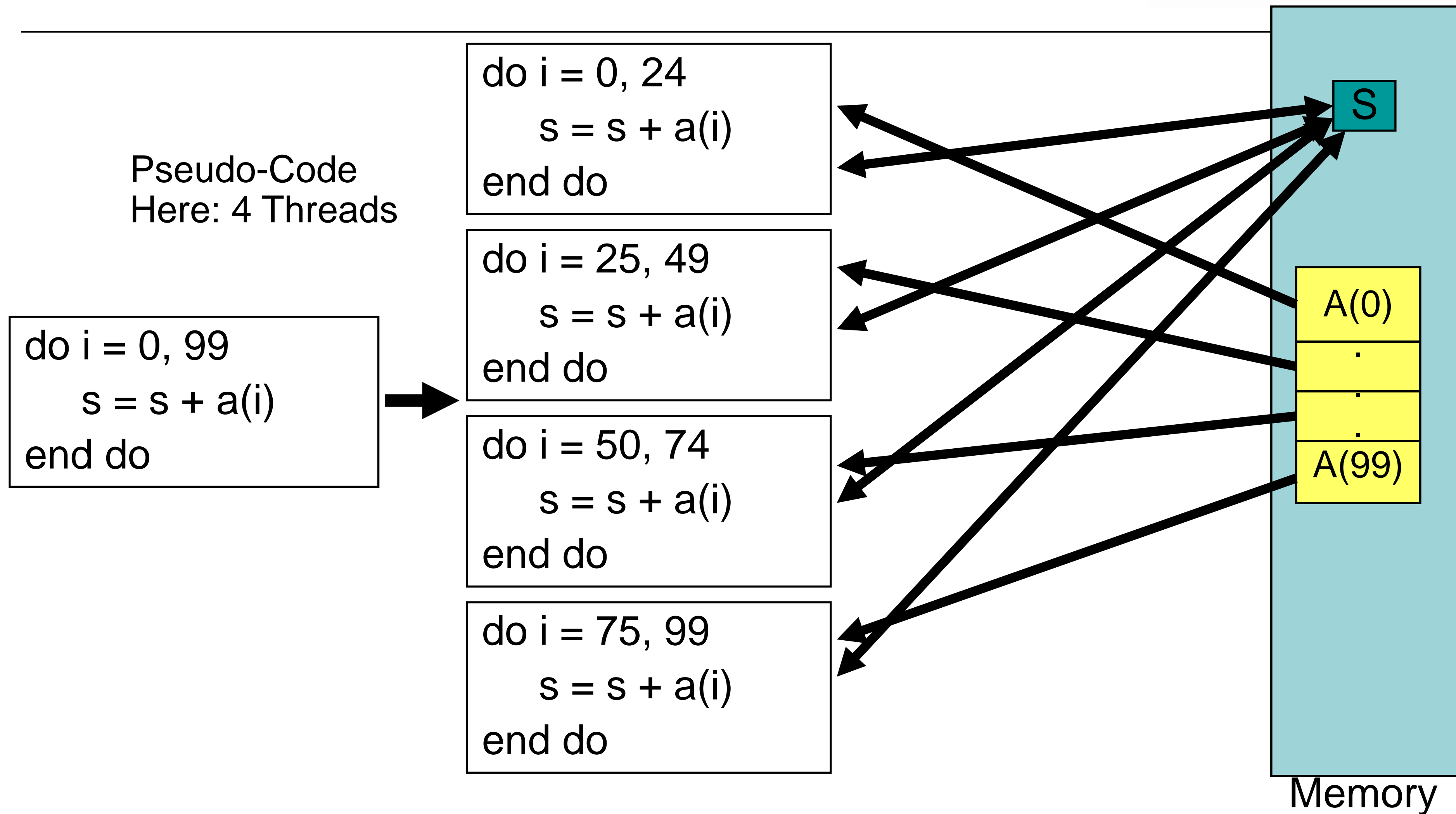
An OpenMP program starts just like a serial program with one thread: the so-called *Master* thread.

At a so-called *Parallel Region* slave threads are spawned, together with the master thread they form a *Team*.

Data may be shared among all threads or be private to a thread.

SC SCIENTIFIC COMPUTING

An attempt at parallelization



SC SCIENTIFIC COMPUTING

Problem, when several threads write to the same memory location

***Data Race*:** If between two synchronization points at least one thread writes to a memory location from which at least one other thread reads, the result is not deterministic (race condition).

Synchronization

- Coordination of parallel execution is required to avoid data races and to ensure correctness of program
- Two forms of synchronization:
  - **Mutual exclusion** – only one thread is guaranteed access to a memory location, the others have to wait!
  - **Event synchronization**, for example a so-called barrier, that is a point where each threads waits for all other threads to arrive.
- Synchronization is expensive: In the time that a synchronization is performed, 100 – 1000 operations can be performed!
- By default, data is shared among threads.

Mutual exclusion with a critical region

TECHNISCHE UNIVERSITÄT DARMSTADT

A *Critical Region* is executed by all threads, but by only one thread simultaneously.

```
int i;
#pragma omp parallel
{
#pragma omp for
for (i = 0; i < 100; i++)
{
#pragma omp critical
   {  s = s + a[i];  }
}
}
```


SC SCIENTIFIC COMPUTING

Improving Parallelism

TECHNISCHE UNIVERSITÄT DARMSTADT

```
#pragma omp parallel
{


#pragma omp for
   for (i = 0; i < 100; i++)
   {
      #pragma omp critical
      {
         s     = s      + a[i];
      }
   }



} // end parallel
```


SCIENTIFIC COMPUTING

Using single construct

TECHNISCHE UNIVERSITÄT DARMSTADT

```
 int nthreads = omp_get_max_threads();
#pragma omp parallel
{
   int tid = omp_get_thread_num();
#pragma omp for
   for (i = 0; i < 100; i++)
   {
      s_priv[tid] = s_priv[tid] + a[i];
   }
#pragma omp single
    for (i = 0; i < nthreads; i++)
         s += spriv[i];
} // end parallel
```

Reduction variables

TECHNISCHE UNIVERSITÄT DARMSTADT

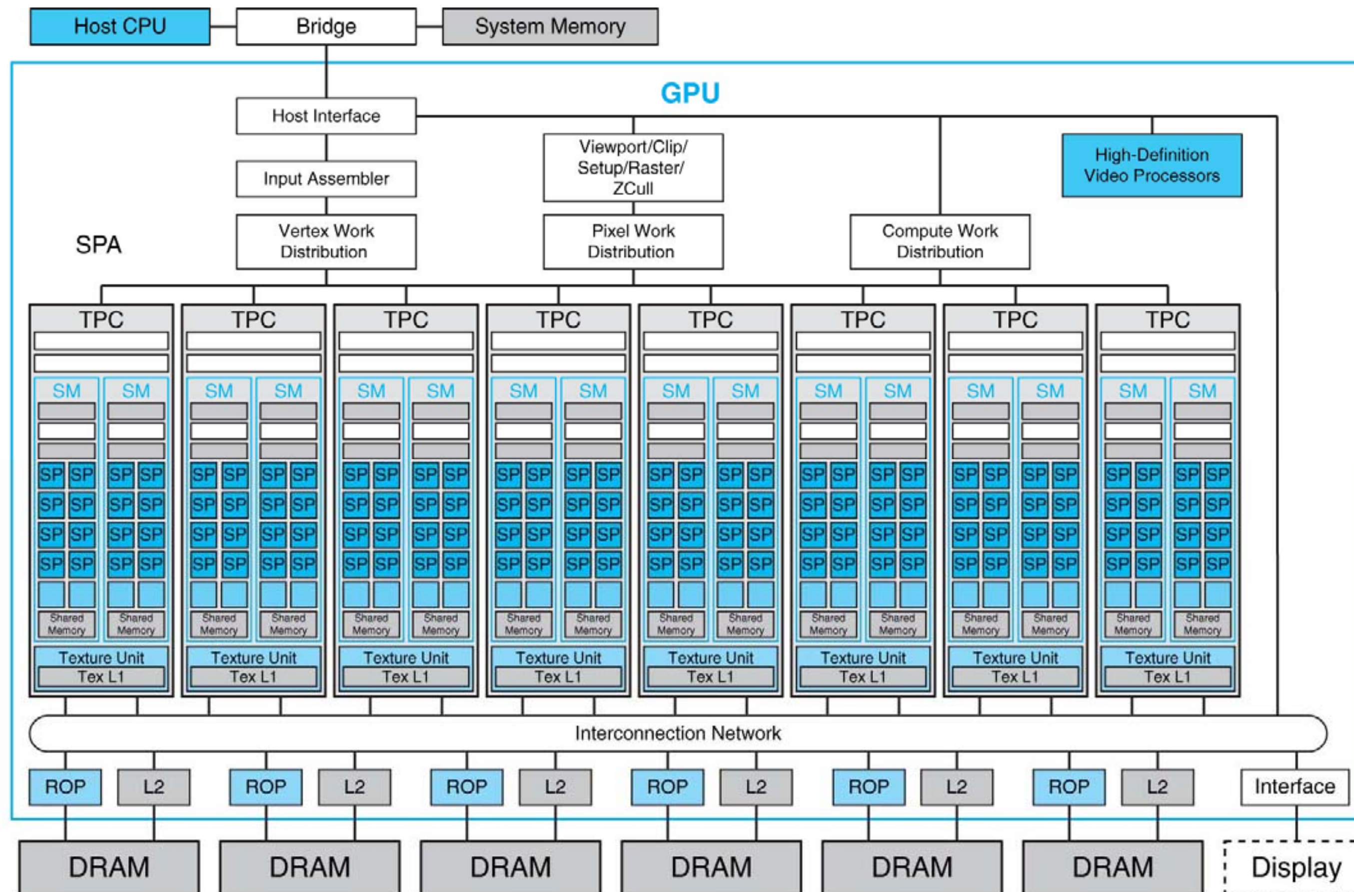
```
#pragma omp parallel
{
#pragma omp for reduction(+:s)
   for (i = 0; i < 100; i++)
   {

          s = s + a[i];

   }
} // end parallel
```

The OPENMP compiler translates the reduction pragma into efficient parallel code (local addition of array entries, summing of local contributions at the end).

SC SCIENTIFIC COMPUTING

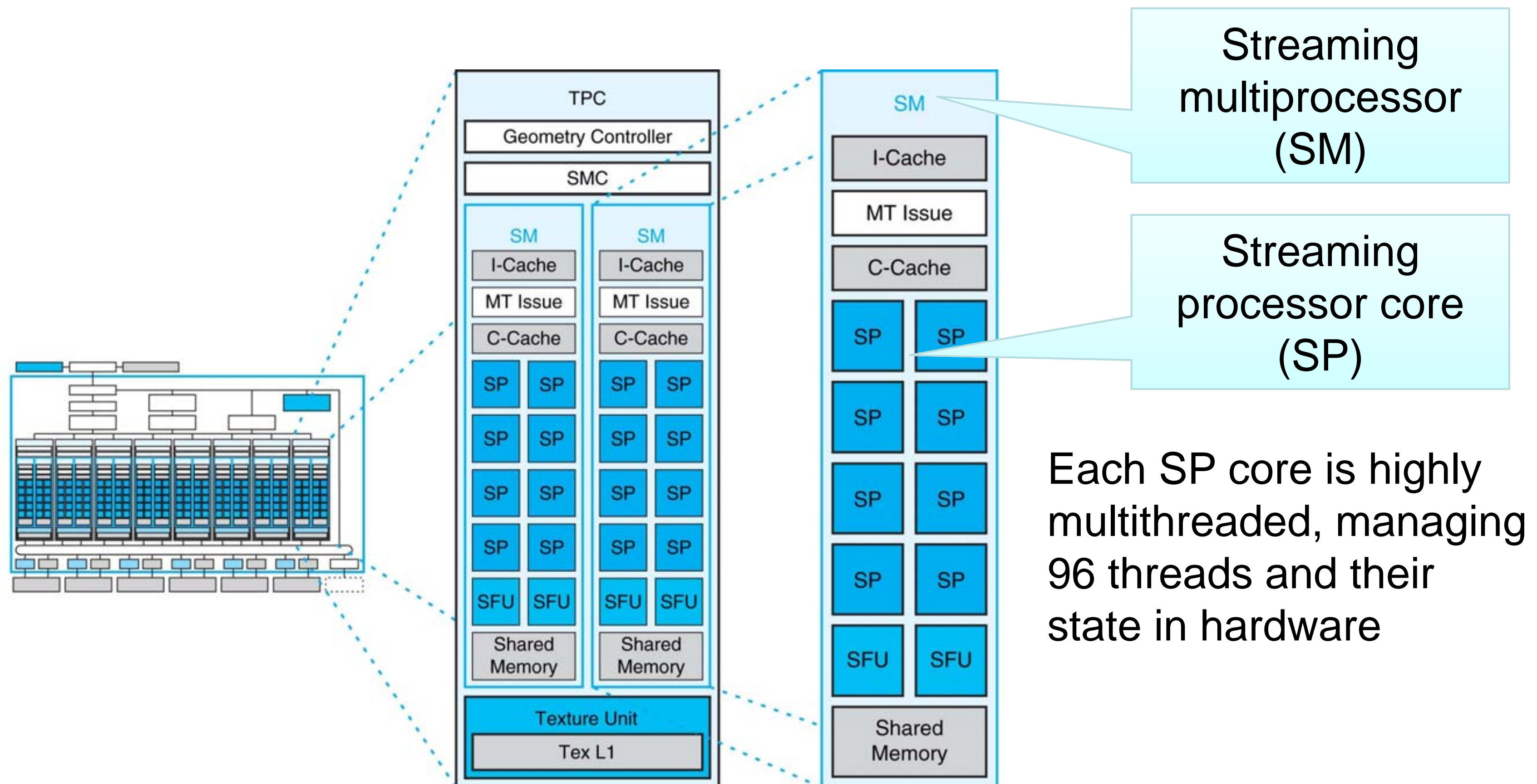
GPGPU PROGRAMMIERUNG

NVIDIA GeForce 8800



Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann

NVIDIA GeForce 8800 (2)



Each SP core is highly multithreaded, managing 96 threads and their state in hardware

Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann

Programming General-Purpose Graphics Processing Units (GPGPUs)

- A variant of the vector programming model.
  - A team of threads executes the same operation.

- As there are many processing units, some thread teams may execute a graphics shader, others may run geometry processing programs.

- A combination of vector processing and multiprogramming: single-instruction multiple-thread (SIMT) architecture

- CUDA = Common uniform device architecture
  - Programming language + runtime library by NVIDIA
  - Vendor-specific!

Computing y = ax + y with serial loop

```
void saxpy_serial(int n, float alpha, float *x, float *y)
{
  int i;
  for (i=0; i<n; i++)
    y[i]= alpha * x[i] + y[i];
}
/* invoke serial saxpy kernel */
saxpy_serial(n, 2.0, x, y);
```

Computing y = ax + y with parallel loop

```
__global__
void saxpy_parallel(int n, float alpha, float *x, float *y)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i<n) y[i]= alpha * x[i] + y[i];
}
/* invoke parallel saxpy kernel with n threads */
/* organized in 256 threads per block */
int nblocks = (n +255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

OpenCL = Open Computing Language

- Open Specification, owned by Khronos group
  http://www.khronos.org/

- Originally initiated by Apple, to avoid vendor lock-in when using GPGPUs.

- For example, GPGPUs by AMD can be programmed only using their internal low-level language or OpenCL.

OpenCL example for SAXPY

Close to a hardware model, low level of abstraction

TECHNISCHE UNIVERSITÄT DARMSTADT

```c
#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;
}
```


SCIENTIFIC COMPUTING

Remembering Steve Jobs

TECHNISCHE UNIVERSITÄT DARMSTADT

> **"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

- OpenACC effort (www.openacc.org)
  - driven by NVIDIA, Cray, and PGI
- Directives, similar to OpenMP, for programming GPGPUs.

Standard approaches for programming high-performance computers

Programming Languages:

- Fortran, C, C++

For multicore processors:

- OpenMP
- Pthreads

For GPGPUs:

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

For clusters (i.e. computers on a network):

- MPI (Message-Passing Interface)

DIE BEDEUTUNG DER SPEICHERARCHITEKTUR

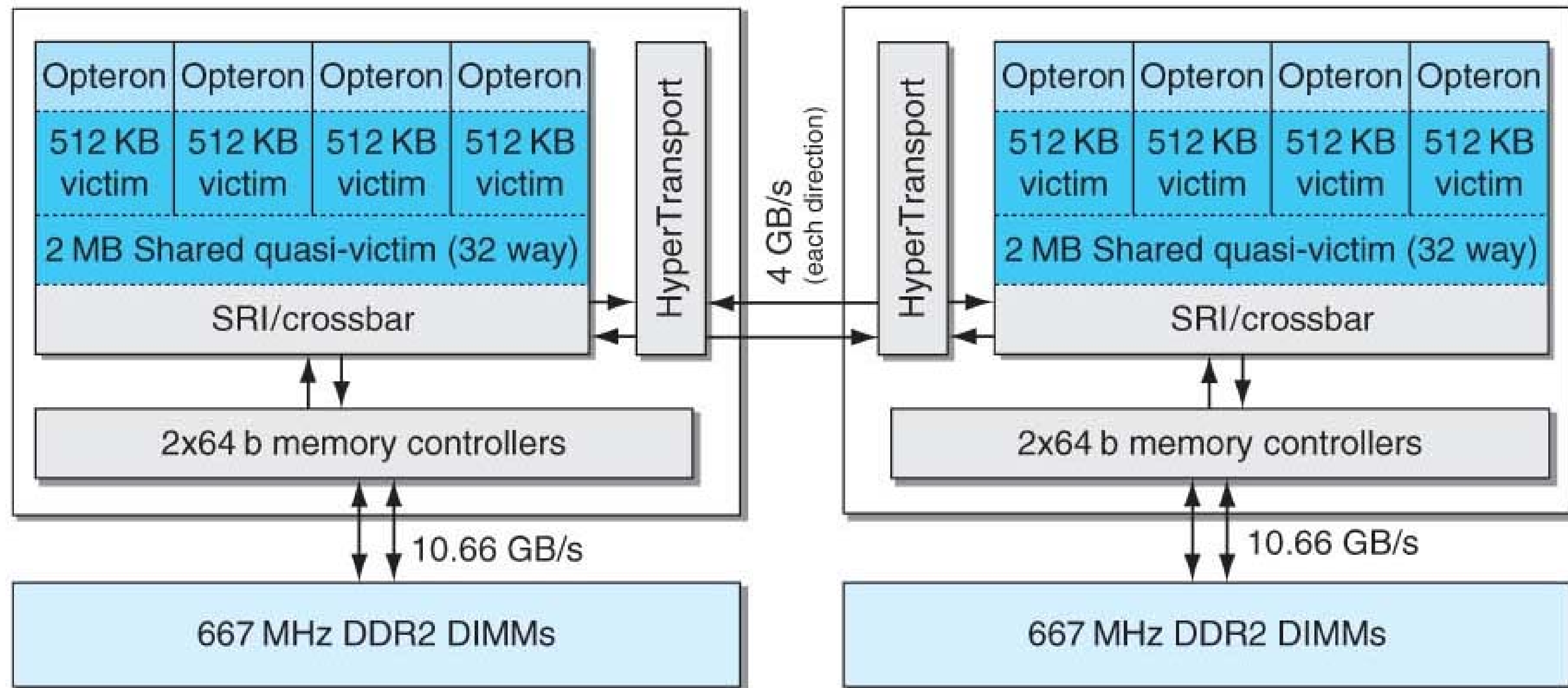
Do not forget the memory

TECHNISCHE UNIVERSITÄT DARMSTADT

- Moderne Chips haben Zwischenspeicher, sog. Caches.
  - Eine Kaskade von Speichern, die, je weiter sie von der CPU weg sind, immer langsamer und größer werden.
  - L1 cache – L2 cache – L3 cache
- Ob die Daten aus dem Hauptspeicher, oder einer prozessornahen cache kommen, hat großen Einfluss auf die Leistung.
- Wenn das Programm keine Leistung aus einem Prozessor herausholt, dann wird es auch auf 1000 Prozessoren ineffizient laufen
- Gerhard Wellein@Uni Erlangen: „1000 * 0 = 0"

SC SCIENTIFIC COMPUTING

AMD Opteron X4 2356 (Barcelona)

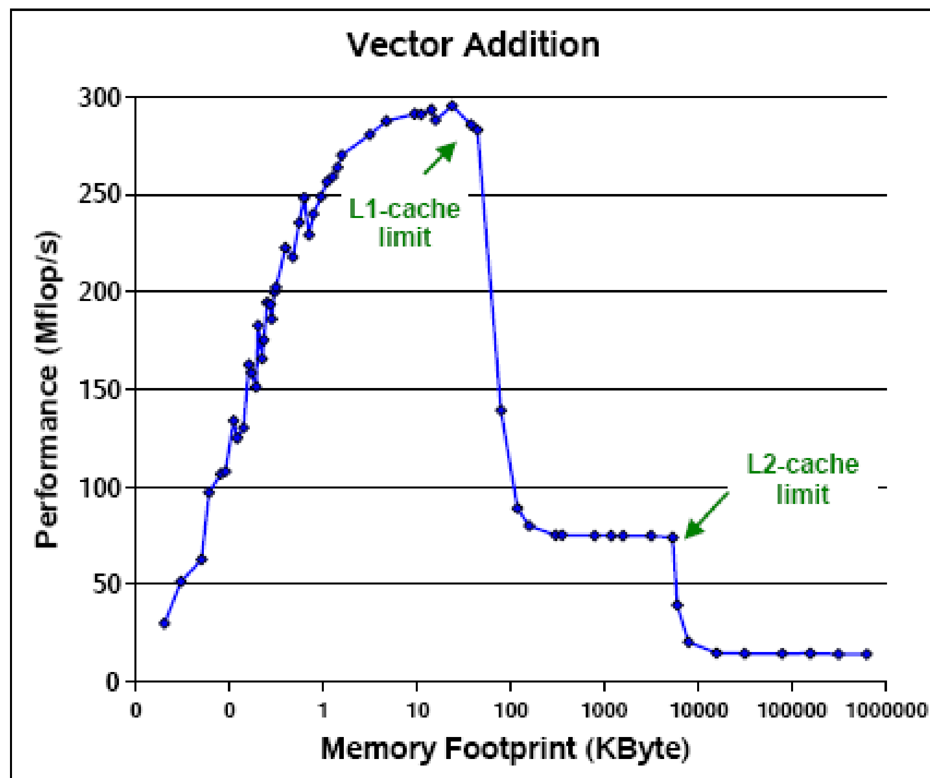


Source: Patterson, Hennessy: Computer Organization & Design, 4th edition, Morgan Kaufmann, 2011

Performance of Vector Addition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for (i=0; i<vlen; i++)
   p[i] = q[i] + r[i];
```



- *This operation is* ***memory bound*** *i.e. there are more memory references than floating point operations*
- *The system realizes close to the theoretical peak performance for* ***this*** *operation (i.e. 1/6 of absolute peak)*
- *Note the start-up effect and the performance drop for larger problems*

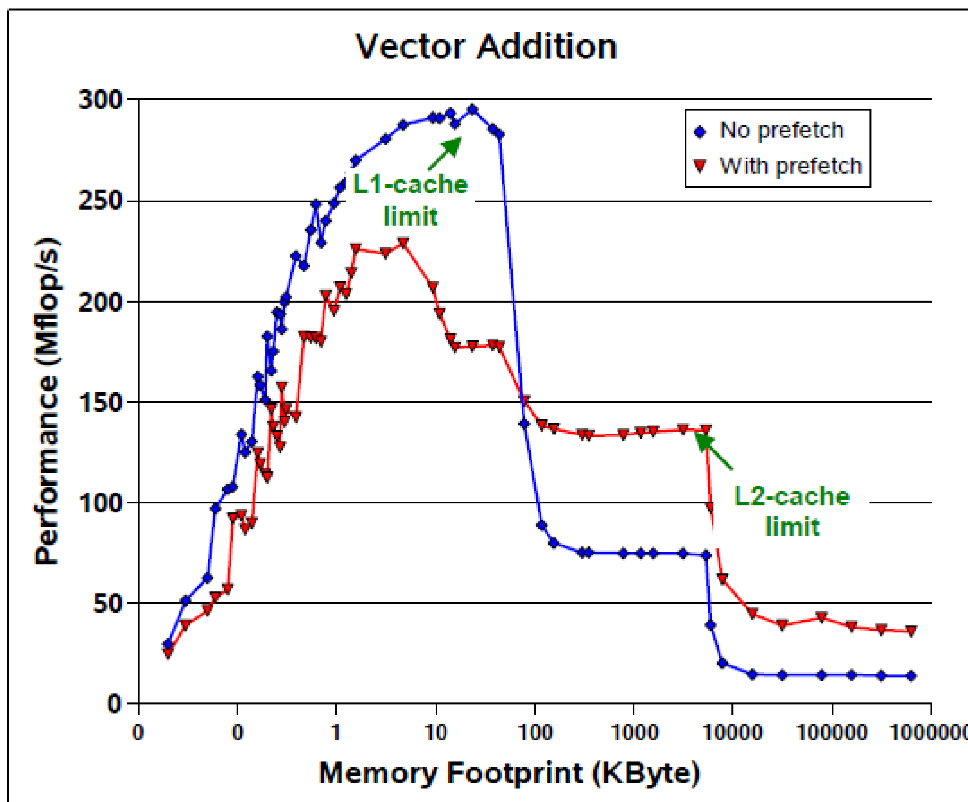
Courtesy of Ruud van der Pas
Sun Microsystems

SF6800 - USIII Cu@900MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1800 Mflop/s

SCIENTIFIC COMPUTING

Prefetch – Vector Addition

```
for (i=0; i<vlen; i++)
   p[i] = q[i] + r[i];
```



- ***Re-compiled with automatic prefetch enabled***
- ***Performance for L1 resident problem sizes is less if prefetch is used***
- ***For larger problem sizes, automatic prefetch gives a significant performance improvement***

SF6800 - USIII Cu@900MHz
L1 cache : 64 KByte
L2 cache : 8 MByte
Peak speed : 1800 Mflop/s

Das Software Loch

TECHNISCHE UNIVERSITÄT DARMSTADT

- Die parallele Programmierung ist intellektuell anspruchsvoll.
- Das niedrige Abstraktionsniveau macht paralleles Programmieren schwierig und verleitet Programmierer dazu, auf eine ganz bestimmte Architektur hin zu programmieren.
  - Wenn dann der alte Code auf eine neue, schnellere Maschine umgezogen wird, kann die Leistung sinken!
- **Das Software Loch (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*

SC SCIENTIFIC COMPUTING

Produktivität versus Rechenleistung

- Traditionell war HPC Programmierung von Rechenleistung getrieben – heroische Programmierer, die das letzte aus „ihrer" Maschine herauskitzelten.
- In der Zukunft ist **Produktivität** mindestens genau so wichtig:
  Wie lange dauert es, bis eine wissenschaftliche Idee ihren Ausdruck findet in einem Programm, das
  - verifiziert ist,
  - dokumentiert ist,
  - wartbar ist und
  - erweiterbar ist.
- Dies sind klassische Fragestellungen des Software Engineering.
- Wir können die Vielfalt an Rechnern, die uns das Moore'sche Gesetzt beschert hat, nutzen, um die Architektur zu wählen, die diese Aufgabe erleichtert.

TECHNISCHE UNIVERSITÄT DARMSTADT

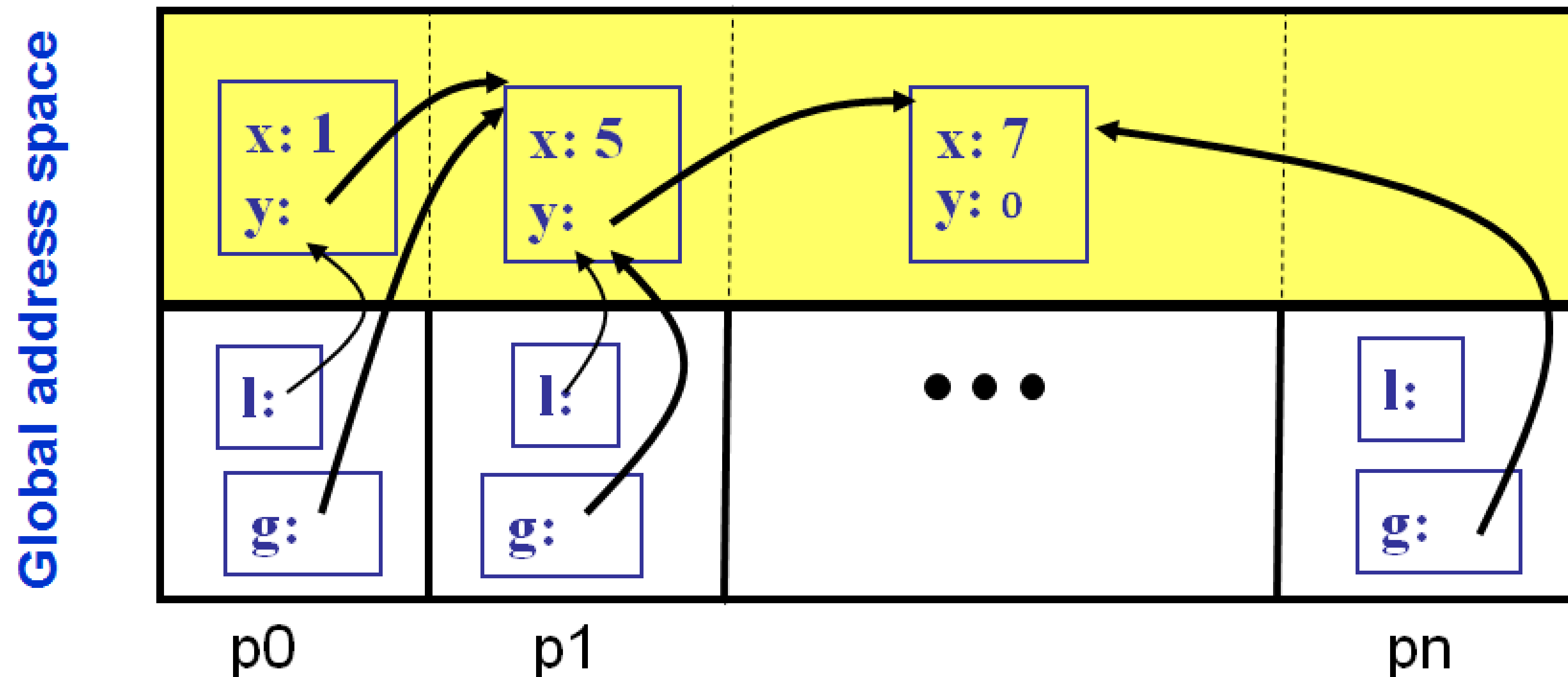
WIE KOMMEN WIR AUS DEM SOFTWARE LOCH HERAUS?

SCIENTIFIC COMPUTING

Lösungsansätze

- Neue Programmiersprachen
  - PGAS
  - Domain-specific languages
- Automatisierte Code Erzeugung
- Automatisches Code Tuning

Partitioned Global Array Languages (PGAS)



- Idee: Speicher ist explizit in lokalen und globalen Speicher aufgeteilt
- Erlaubt schwächere Synchronisationsannahmen als bei OpenMP, so dass effektive Implementierung auch auf großen Systemen möglich.

Domain-Specific Languages - gPROMS

TECHNISCHE UNIVERSITÄT DARMSTADT

```
EQUATION
  $dist = velo;
  $time = 1.0;
  $velo = 4/pi*atan(accel) - alpha*velo^2;
END
```

- Eine Modellbeschreibung in gPROMS, einer in der Verfahrenstechnik genutzten „Programmiersprache“.
- $ = d /dt (Zeitableitung).
- „=„ bedeutet Gleichheit, nicht Zuweisung.

- Der Programmierer muss sich um Datenstrukturen, numerische Algorithmen (z.B. für Zeitintegration, Lösung nichtlinearer Gleichungssysteme einschl. hierfür benötigter Ableitungen) nicht kümmern.

SC SCIENTIFIC COMPUTING

Domain-Specific Languages - FLAME

Step	Annotated Algorithm: $A := \text{CHOL_UNB_VAR3}(A)$
1a	$\{A = \hat{A}\}$
4	Partition $A \rightarrow \left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right)$, $L \rightarrow \left(\begin{array}{cc} L_{TL} & 0 \\ \hline L_{BL} & L_{BR}\end{array}\right)$ where A_{TL} and L_{TL} are 0×0
2	$\left\{\left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right) = \left(\begin{array}{cc} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T\end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T\right\}$
3	while ... do
2,3	$\left\{\left(\left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right) = \left(\begin{array}{cc} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T\end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T\right) \wedge (m(A_{TL}) < m(A))\right\}$
5a	Repartition $\left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right) \rightarrow \left(\begin{array}{ccc} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22}\end{array}\right)$, $\left(\begin{array}{cc} L_{TL} & 0 \\ \hline L_{BL} & L_{BR}\end{array}\right) \rightarrow \left(\begin{array}{ccc} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22}\end{array}\right)$ where α_{11} and λ_{11} are scalars
6	$\left\{\left(\begin{array}{ccc} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22}\end{array}\right) = \left(\begin{array}{ccc} L_{00} & \star & \star \\ \hline l_{10}^T & \hat{\alpha}_{11} - l_{10}^T l_{10} & \star \\ \hline L_{20} & \hat{a}_{21} - L_{20}l_{10} & \hat{A}_{22} - L_{20}L_{20}^T\end{array}\right) \wedge \hat{A}_{00} = L_{00}L_{00}^T\right\}$
8	$\alpha_{11} := \sqrt{\alpha_{11}}$ $a_{21} := a_{21}/\alpha_{11}$ $A_{22} := A_{22} - a_{21}a_{21}^T$
5b	Continue with $\left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right) \leftarrow \left(\begin{array}{ccc} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22}\end{array}\right)$, $\left(\begin{array}{cc} L_{TL} & 0 \\ \hline L_{BL} & L_{BR}\end{array}\right) \leftarrow \left(\begin{array}{ccc} L_{00} & 0 & 0 \\ \hline l_{10}^T & \lambda_{11} & 0 \\ \hline L_{20} & l_{21} & L_{22}\end{array}\right)$
7	$\left\{\left(\begin{array}{ccc} A_{00} & \star & \star \\ \hline a_{10}^T & \alpha_{11} & \star \\ \hline A_{20} & a_{21} & A_{22}\end{array}\right) = \left(\begin{array}{ccc} L_{00} & \star & \star \\ l_{10}^T & \lambda_{11} & \star \\ \hline L_{20} & l_{21} & \hat{A}_{22} - L_{20}L_{20}^T - l_{21}l_{21}^T\end{array}\right) \wedge \left(\begin{array}{cc} \hat{A}_{00} & \star \\ \hat{a}_{21} & \hat{A}_{22}\end{array}\right) = \left(\begin{array}{cc} L_{00}L_{00}^T & \star \\ \lambda_{11}l_{21} & l_{21}l_{21} + \lambda_{11}^2\end{array}\right)\right\}$
2	$\left\{\left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right) = \left(\begin{array}{cc} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T\end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T\right\}$
	endwhile
2,3	$\left\{\left(\left(\begin{array}{cc} A_{TL} & \star \\ \hline A_{BL} & A_{BR}\end{array}\right) = \left(\begin{array}{cc} \hat{L}_{TL} & \star \\ \hline \hat{L}_{BL} & \hat{A}_{BR} - L_{BL}L_{BL}^T\end{array}\right) \wedge \hat{A}_{TL} = L_{TL}L_{TL}^T\right) \wedge \neg(m(A_{TL}) < m(A))\right\}$
1b	$\{A = L \wedge \hat{A} = LL^T\}$

- FLAME = Formal Linear Algebra Methods Environment.
- Entwickelt von Robert van de Geijn, University of Texas at Austin.
- Die formale Spezifikation entspricht dem „Denke über Matrizen in Blöcken“-Ansatz.
- **Invarianten stellen Korrektheit sicher** (Hoare Logic).
- Hieraus wird automatisch paralleler Code generiert, für eine Vielzahl von Plattformen.

TECHNISCHE UNIVERSITÄT DARMSTADT

SCIENTIFIC COMPUTING

Automatisierte Code Generierung

- **Wenn genügend Wissen über die Problemdomäne bekannt ist, kann Code Generierung automatisiert werden.**
- Beispiel Automatisches Differenzieren (AD)
  - AD Werkzeuge erzeugen um aus einem Code, der „f(x)“ berechnet, einen neuen, der „df/dx“ berechnet.
  - Damit wird in der Numerik aus einem Simulationscode ein Designwerkzeug!
  - Die Assoziativität der Kettenregel erschließt hier neue Möglichkeiten für paralleles Rechnen!
- Beispiel Geometrische Algebra (GA)
  - Galoop compiler (Geometric Algebra Algorithms Optimizer, Dr. Hildenbrand, TU Darmstadt)

SC SCIENTIFIC COMPUTING

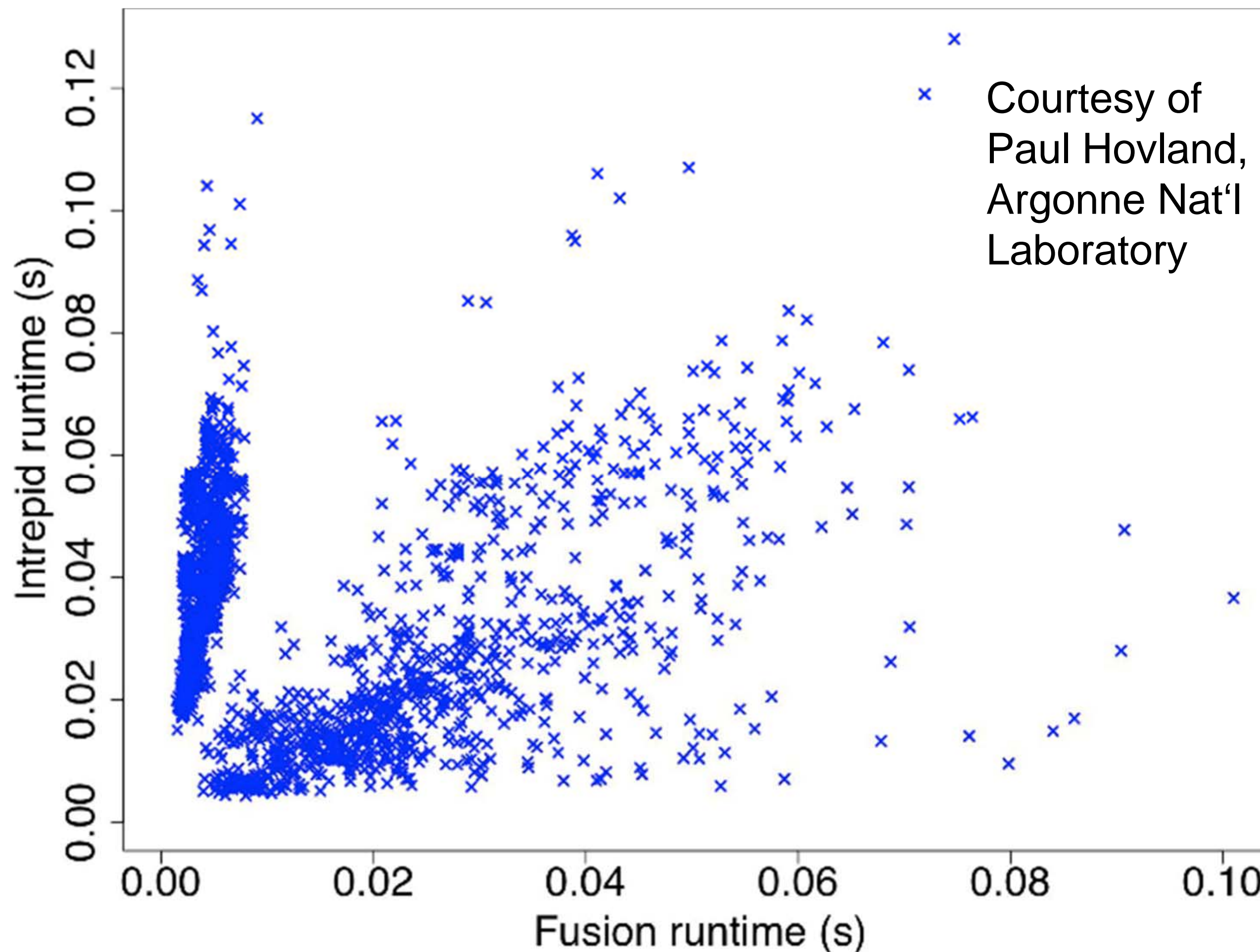
Automatisches Code Tuning

- Viele Algorithmen können konfiguriert werden, um einen Code z.B. auf die Größe von Speichercaches „einzustellen“.
- Traditionell wurde dies durch Testläufe und von Hand gemacht.
- Die Anzahl der Parameter ist aber groß, und Auswirkung auf Leistung sowie Abhängigkeiten untereinander sind sehr von der Architektur abhängig.

SCIENTIFIC COMPUTING

Kernel of the NEK5000 Spectral Element Code

TECHNISCHE UNIVERSITÄT DARMSTADT



- Performance for a large number of randomly chosen tuning parameters (e.g., unroll factor, loop order)
- Fusion is an Intel cluster, see http://www.lcrc.anl.gov
- Intrepid is an IBM Blue Gene System, see http://www-stage.alcf.anl.gov/intrepid

Automatisches Code Tuning …

- **Anpassung der Codes durch gezielte Optimierungsansätze**

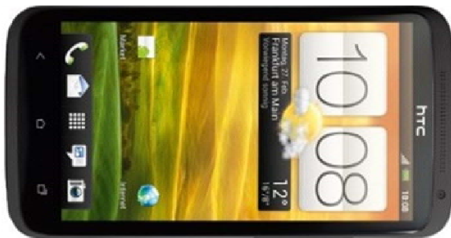
  siehe z.B. An Experimental Study of Global and Local Search Algorithms in Empirical Performance Tuning, Prasanna Balaprakash, Stefan M. Wild, and Paul D. Hovland, http://www.mcs.anl.gov/uploads/cels/papers/P1995-0112.pdf

- Beispiele: PhiPAC, ATLAS, FFTW, Spiral, OSKI, GotoBLAS

Zusammenfassung

TECHNISCHE UNIVERSITÄT DARMSTADT

- Software ist der Schlüssel für Leistung, Produktivität und sinnvolle Nutzung von Hochleistungsrechnern.
- Das Moore'sche Gesetz wird uns weiter interessante Architekturen bescheren.
- Wenn wir also über Algorithmen – Software – Hardware nachdenken, wird Software immer mehr zur Schlüsseltechnologie.
- Der Informatik kommt eine Schlüsselrolle dabei zu, die Möglichkeiten des Hochleistungsrechnens in wissenschaftliche Fortschritte umzusetzen.
- Es gibt keine seriellen Computer mehr!



SC SCIENTIFIC COMPUTING

Think Parallel!

10^{12} neurons @ 1 KHz = 1 PetaOp/s

Courtesy of David Keyes, KAUST