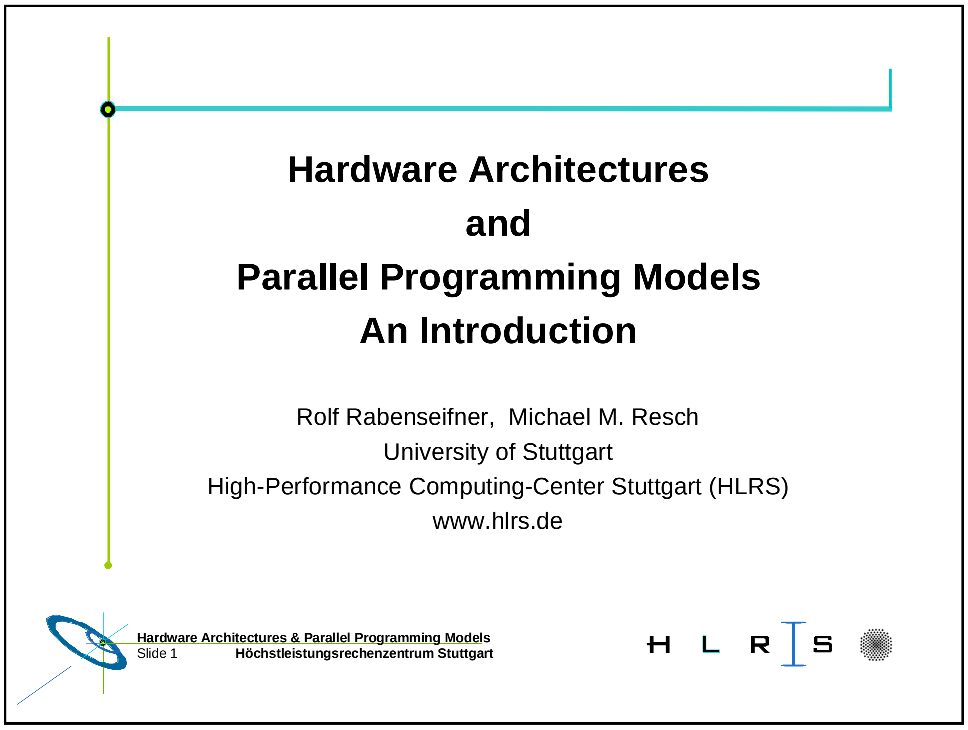
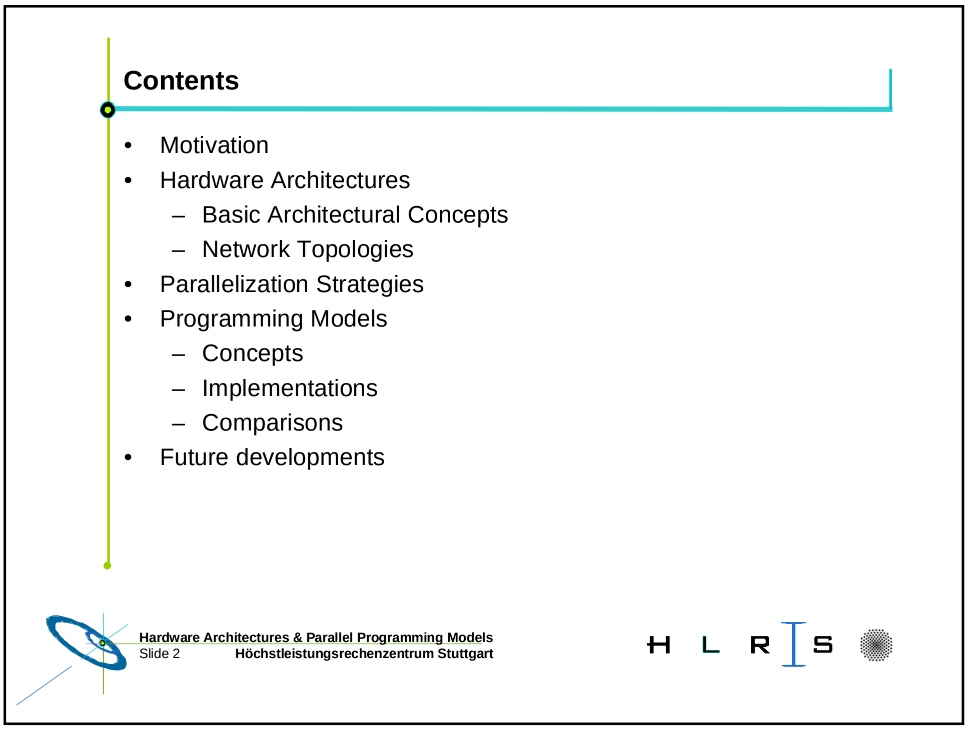


Hardware Architectures and Parallel Programming Models An Introduction

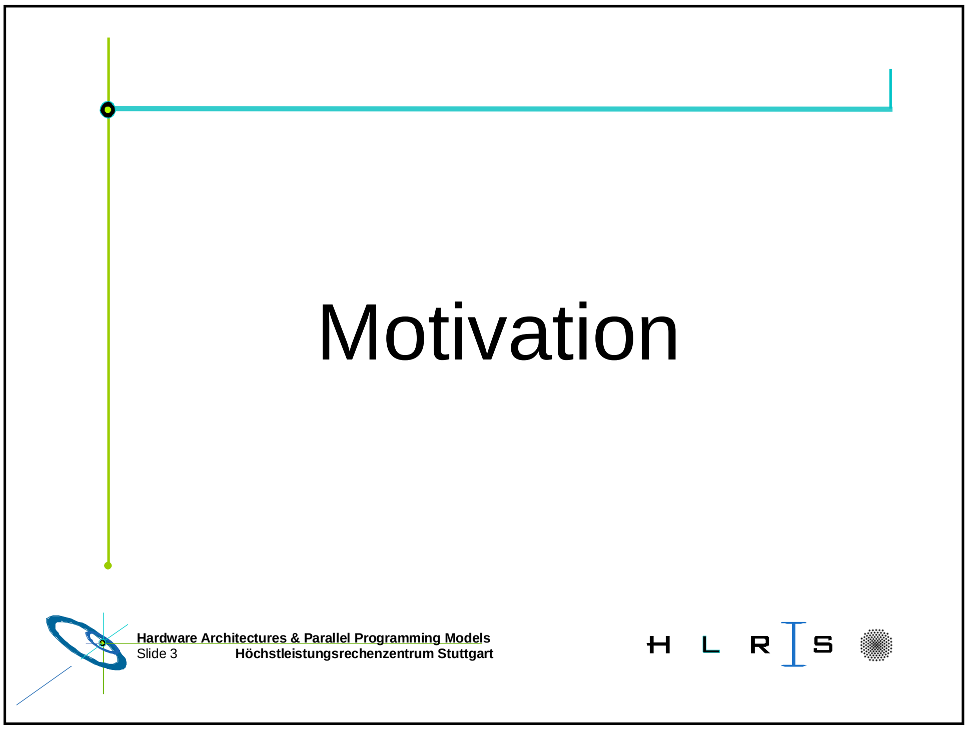
Rolf Rabenseifner, Michael M. Resch
University of Stuttgart
High-Performance Computing-Center Stuttgart (HLRS)
www.hlrs.de

H L R S



Contents

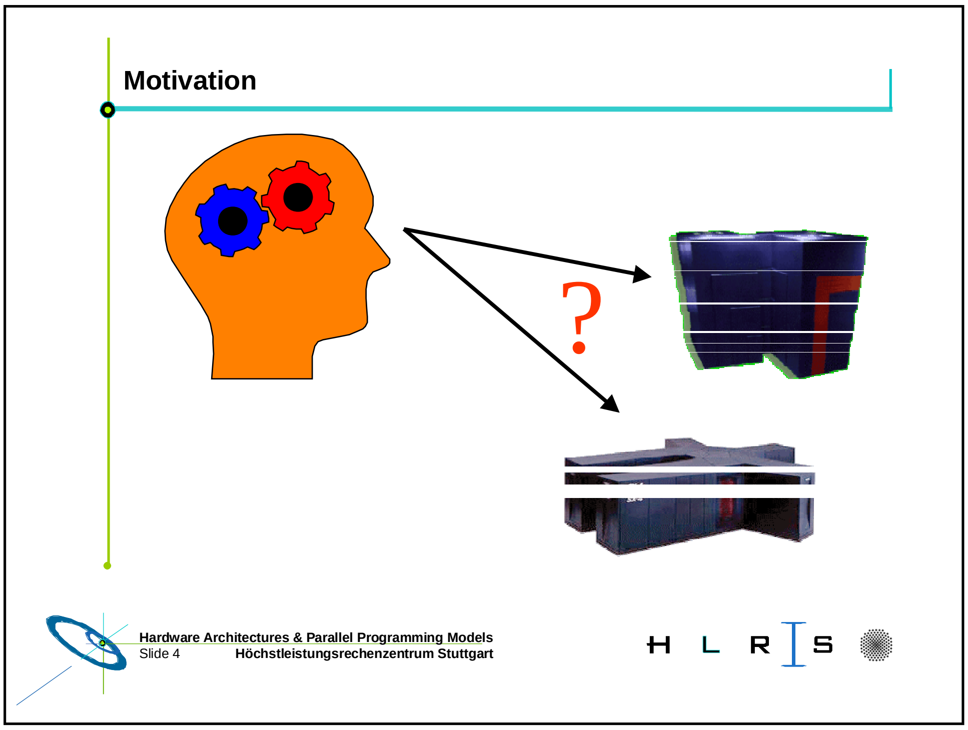
H L R S



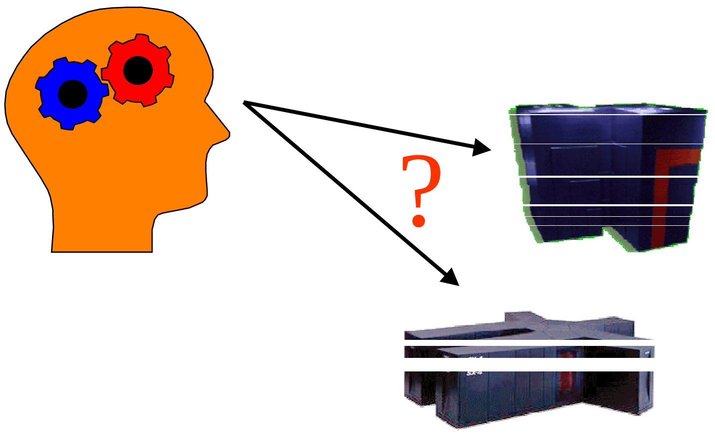
Motivation

Hardware Architectures & Parallel Programming Models
Slide 3 Höchstleistungsrechenzentrum Stuttgart

H L R S



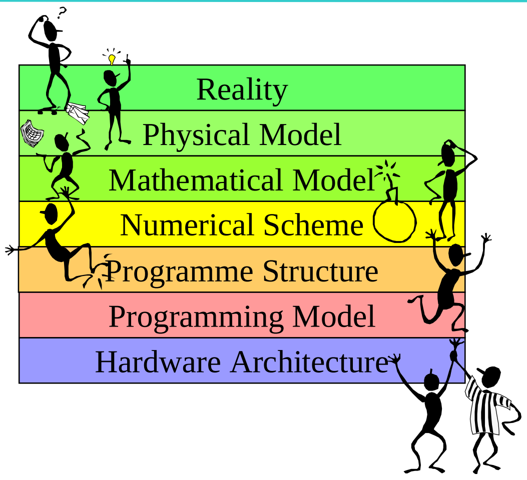
Motivation



Hardware Architectures & Parallel Programming Models
Slide 4 Höchstleistungsrechenzentrum Stuttgart

H L R S

Abstract Model



Parallel Compiler

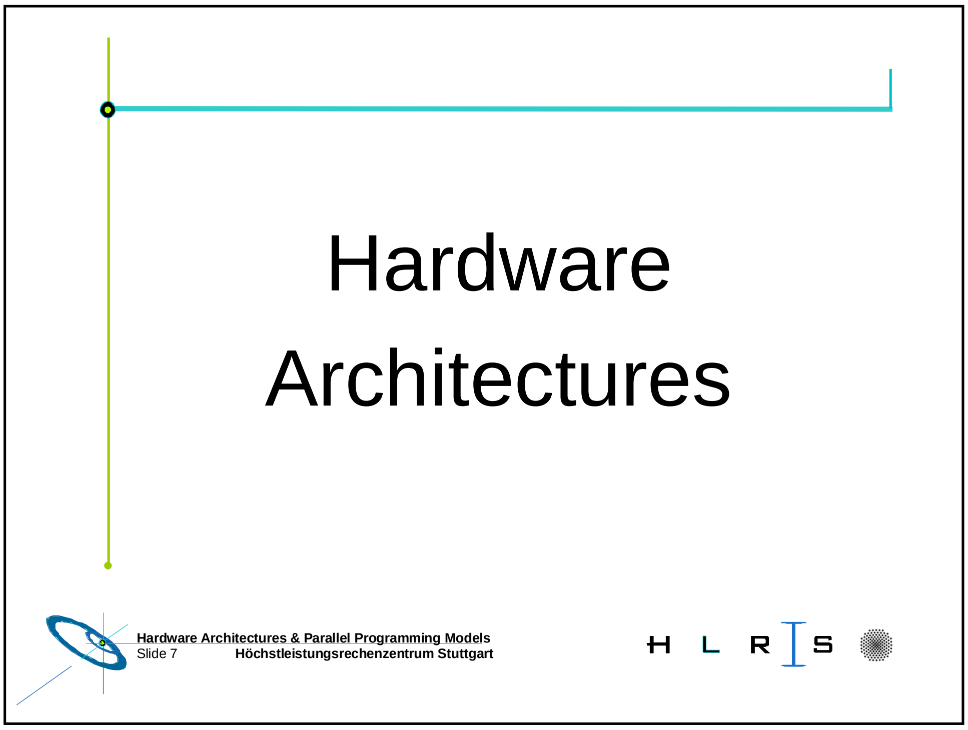
- Why can't I just say

  f90 -Parallel mycode.f

  or

  cc -Parallel mycode.c

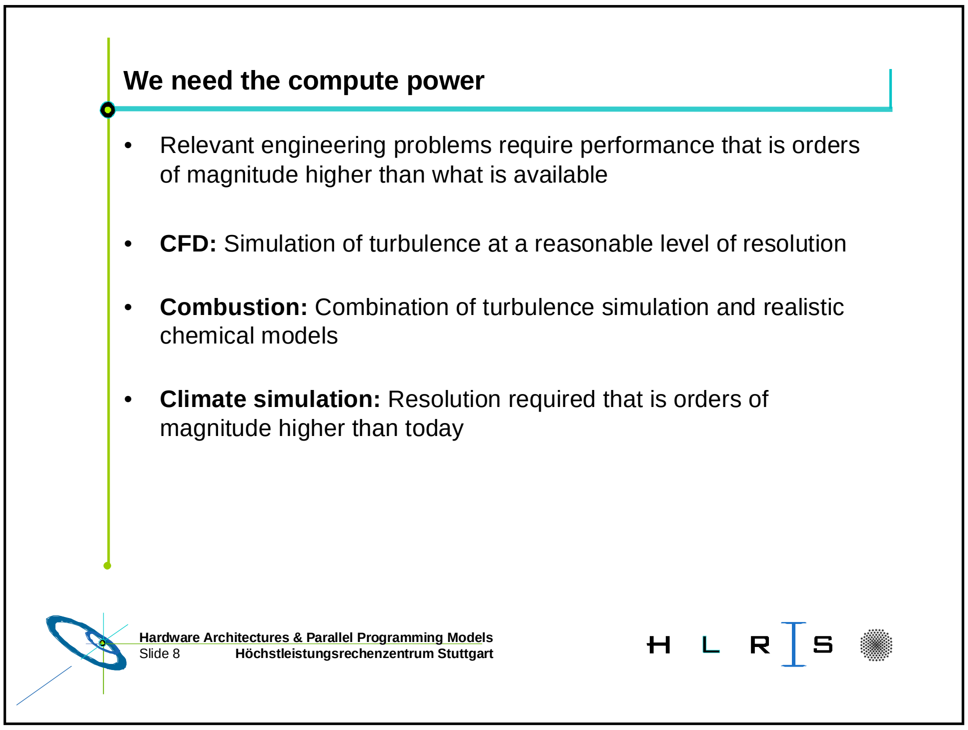
  and everything works fine?



Hardware Architectures

Hardware Architectures & Parallel Programming Models
Slide 7 Höchstleistungsrechenzentrum Stuttgart

H L R S



We need the compute power

- Relevant engineering problems require performance that is orders of magnitude higher than what is available
- **CFD:** Simulation of turbulence at a reasonable level of resolution
- **Combustion:** Combination of turbulence simulation and realistic chemical models
- **Climate simulation:** Resolution required that is orders of magnitude higher than today

Hardware Architectures & Parallel Programming Models
Slide 8 Höchstleistungsrechenzentrum Stuttgart

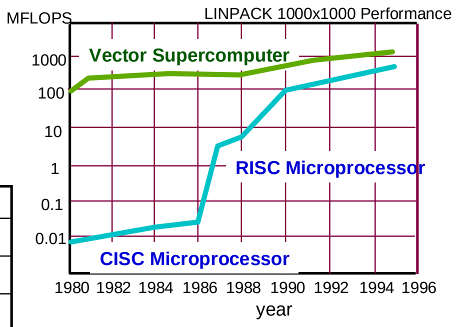
H L R S

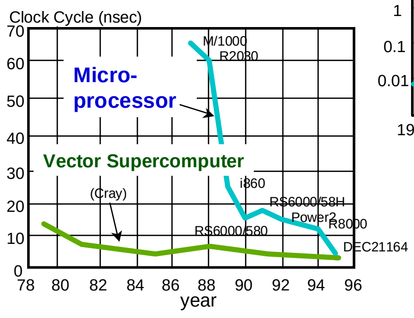
We can not go on like this

- The physical limits of scalar processors are visible
- Clock-rate can no more grow orders of magnitude
- Fast hardware (e.g. ECL or GaAs) has a high power consumption, therefore the potential for higher integration is limited
- High clock-rate needs high density due to the restriction of signal speed by the speed of light

Progress of microprocessor

Information provided by HITACHI



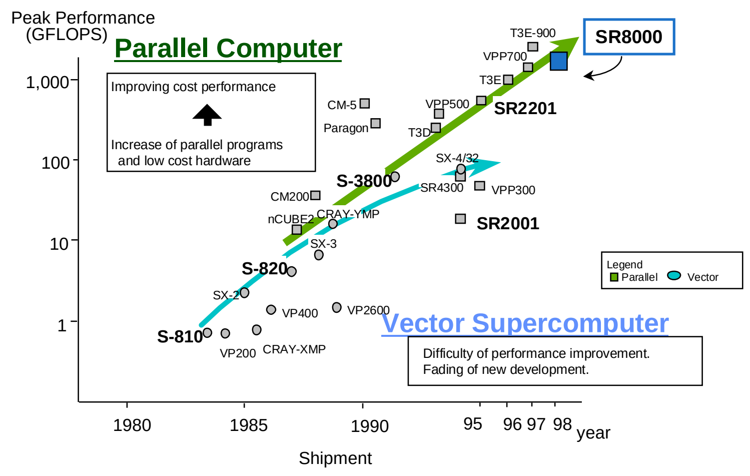


Why not vector computers anymore

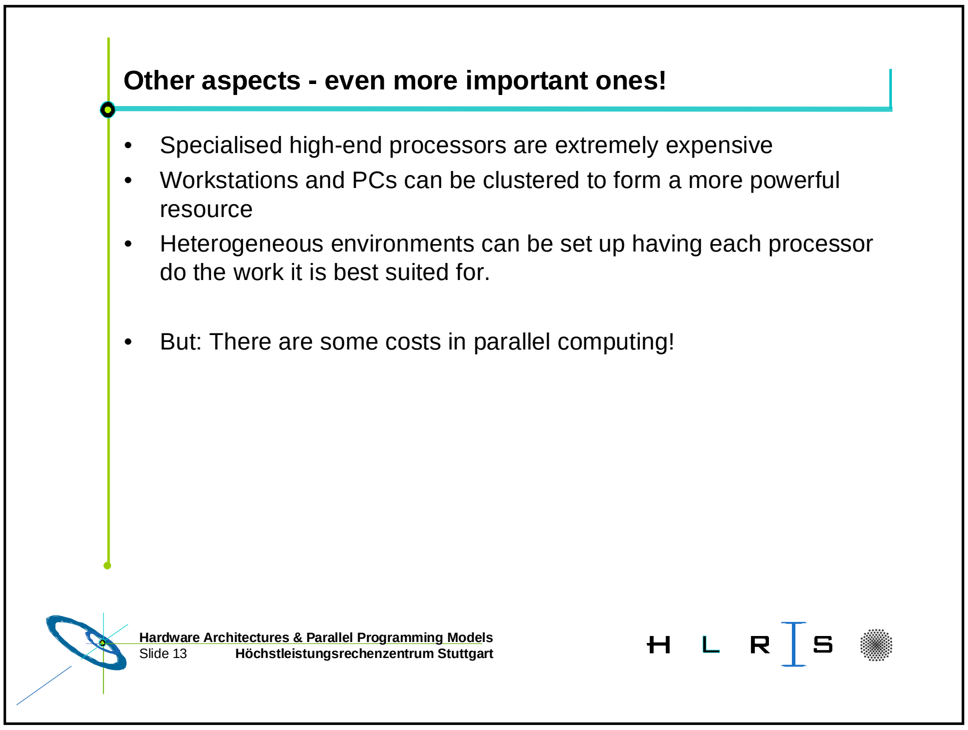
- Higher segmentation not useful for general program and data structures
- Higher segmentation increases pipeline-startup
- Performance improvement needs higher clock rates

Hardware Architectures & Parallel Programming Models
Slide 11 Höchstleistungsrechenzentrum Stuttgart

Evolution of supercomputers



Hardware Architectures & Parallel Programming Models
Slide 12 Höchstleistungsrechenzentrum Stuttgart



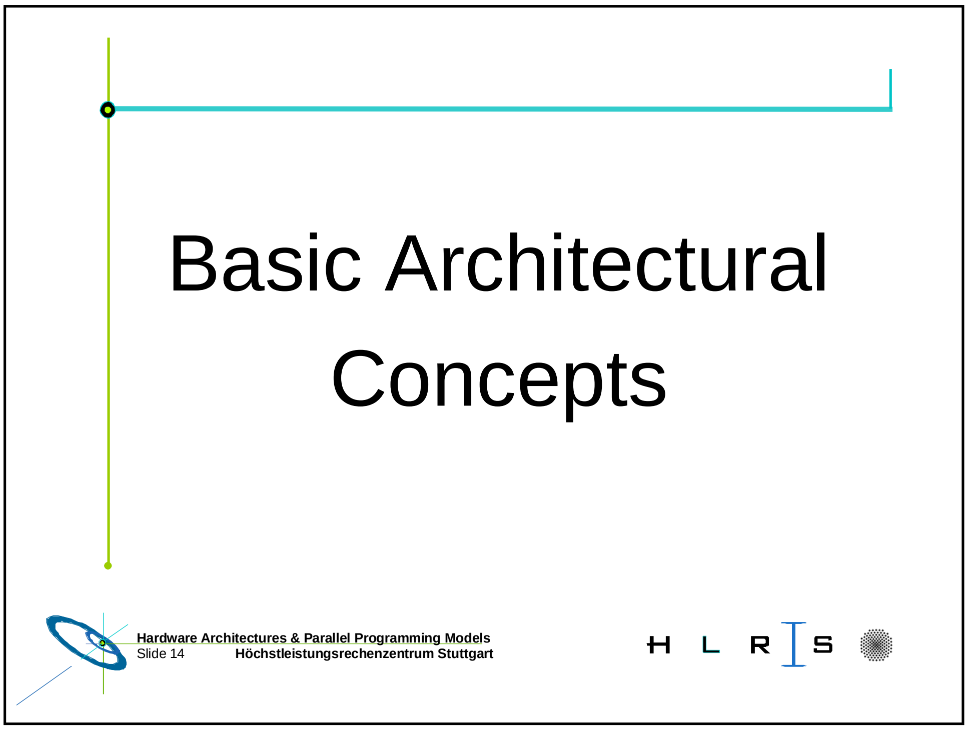
Other aspects - even more important ones!

- Specialised high-end processors are extremely expensive
- Workstations and PCs can be clustered to form a more powerful resource
- Heterogeneous environments can be set up having each processor do the work it is best suited for.

- But: There are some costs in parallel computing!

Hardware Architectures & Parallel Programming Models
Slide 13 Höchstleistungsrechenzentrum Stuttgart

H L R S



Basic Architectural Concepts

Hardware Architectures & Parallel Programming Models
Slide 14 Höchstleistungsrechenzentrum Stuttgart

H L R S

Old and new concepts

Processing concepts:

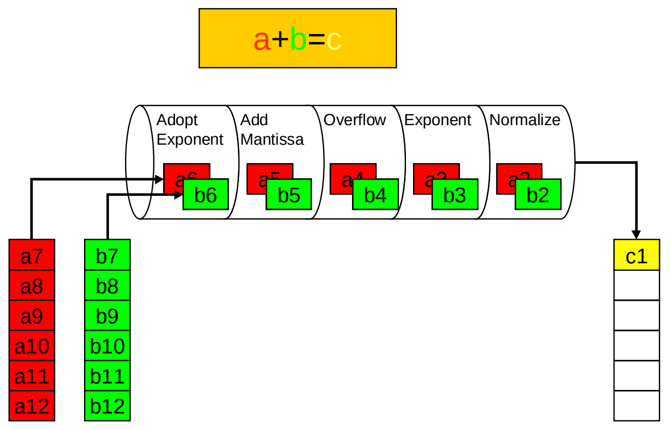
- Pipelining -> that is just vector computing
- Functional Parallelism -> modern processor technology
- Multithreading
- Array-Processing
- Multiprocessors (strongly coupled) -> Shared memory
- Multicomputers (weakly coupled) -> Distributed memory

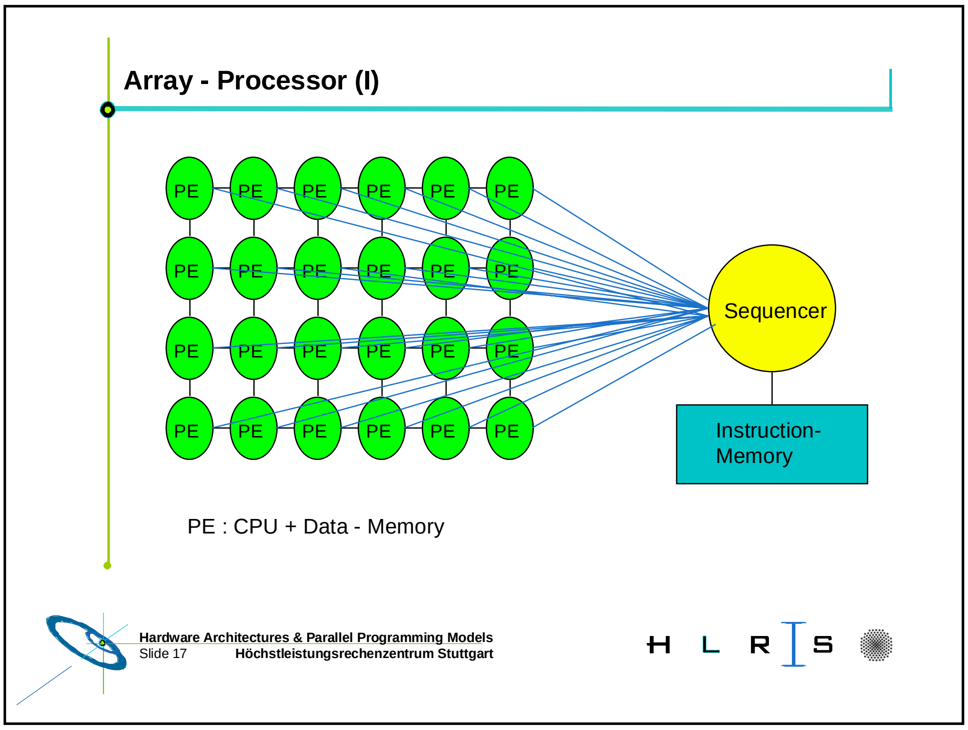
Memory access concepts:

- Cache based
- Vector access via several memory banks
- Pre-load, pre-fetch

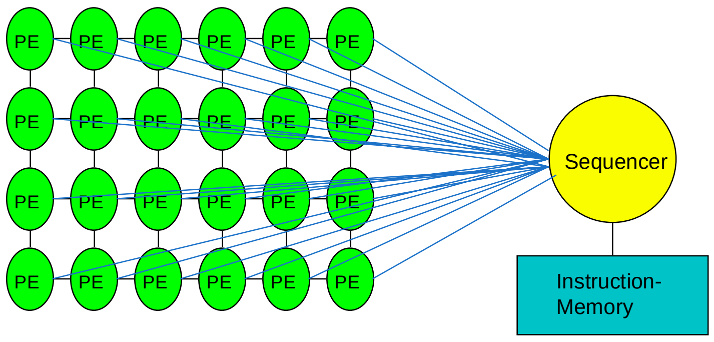
—> MFLOP/s performance **and** MB/s or Mword/s memory bandwidth

Pipelining

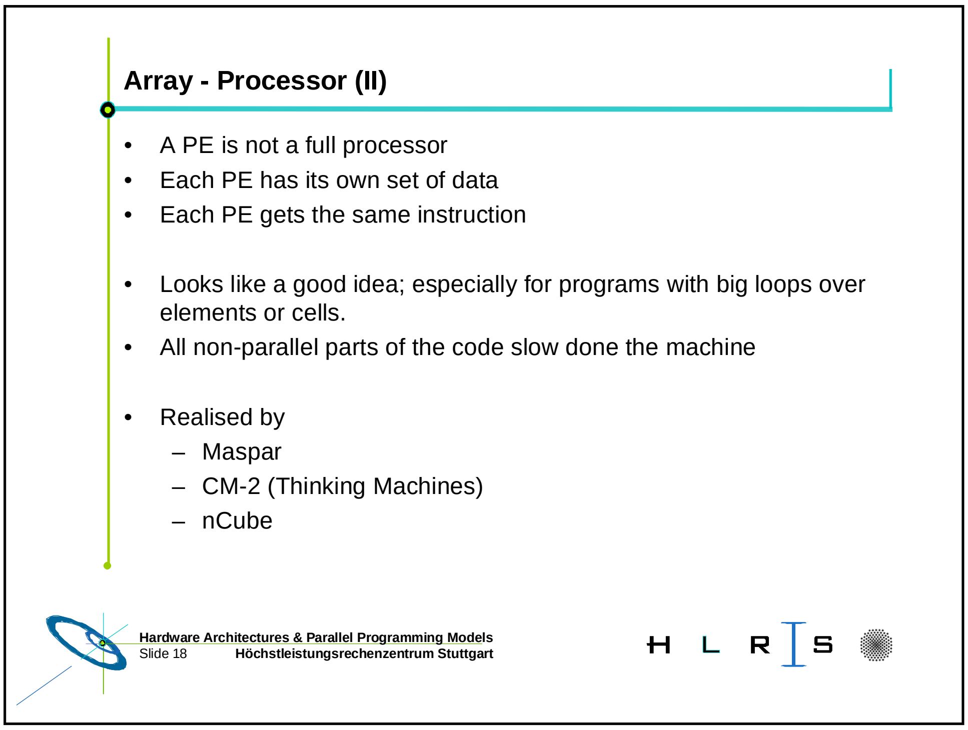




Array - Processor (I)



PE : CPU + Data - Memory



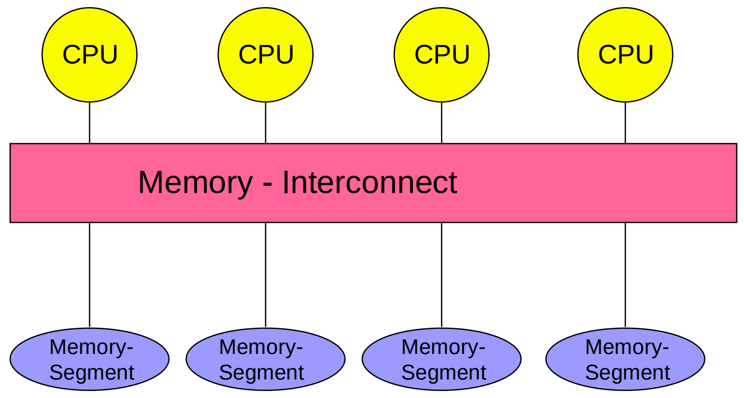
Array - Processor (II)

- A PE is not a full processor
- Each PE has its own set of data
- Each PE gets the same instruction

- Looks like a good idea; especially for programs with big loops over elements or cells.
- All non-parallel parts of the code slow done the machine

- Realised by
  - Maspar
  - CM-2 (Thinking Machines)
  - nCube

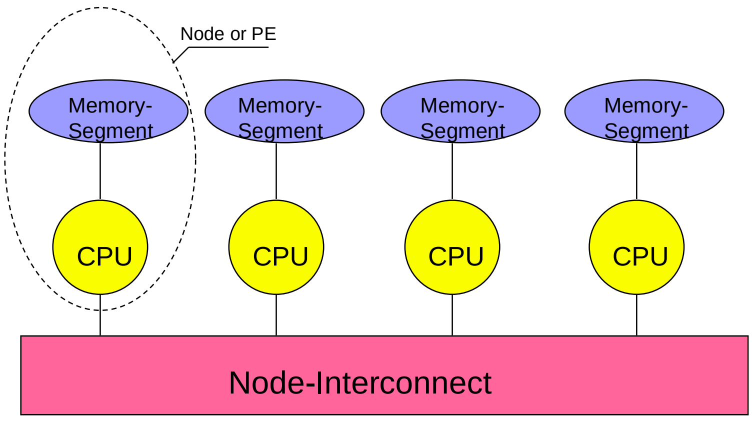
Multiprocessor - shared memory



Multiprocessor

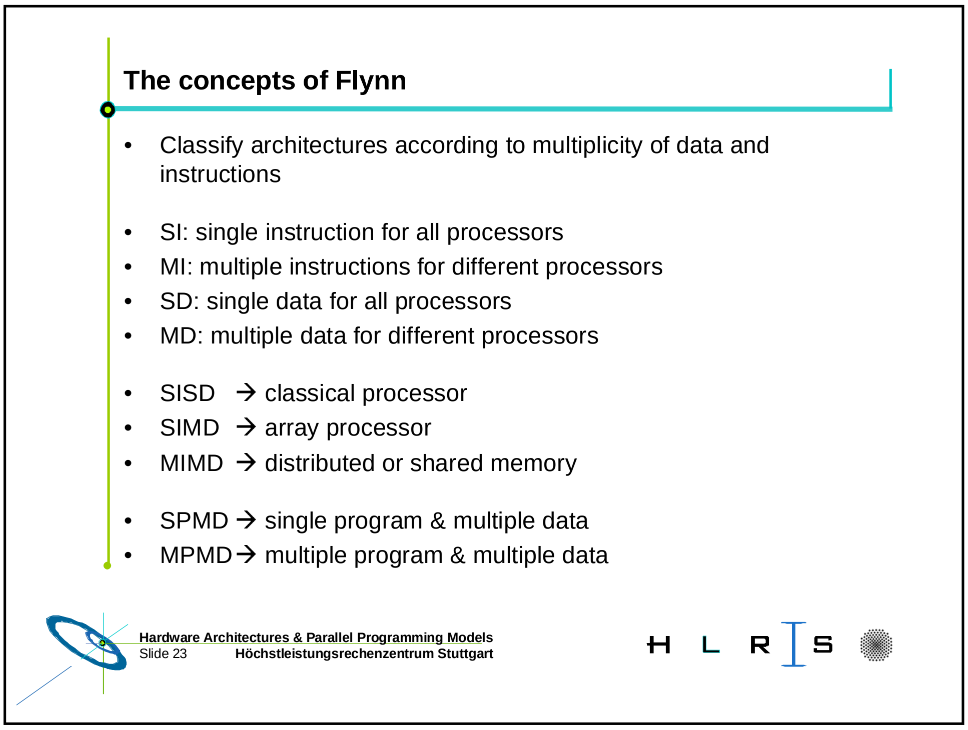
- A number of processors is coupled to a number of memory banks by a fast network
- Each CPU has the same access speed to each memory bank
- This concept is often referred to as **Uniform Memory Access (UMA)**
- The bottleneck may be the network

Multicomputer - distributed memory (I)



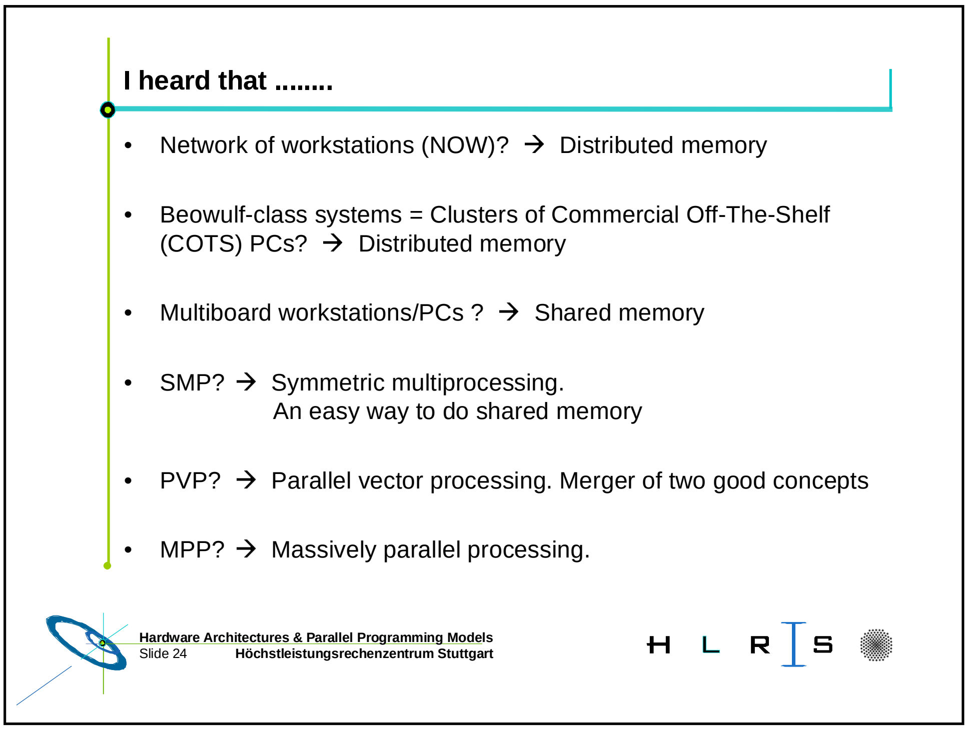
Multicomputer - distributed memory (II)

- A number of full processors with memory is coupled by a fast network
- Each CPU has fast access to its own memory but slower access to other CPU's memories
- This concept is often referred to as **Non-Uniform memory Access (NUMA)**
- Again the network may become a bottleneck



The concepts of Flynn

- Classify architectures according to multiplicity of data and instructions
- SI: single instruction for all processors
- MI: multiple instructions for different processors
- SD: single data for all processors
- MD: multiple data for different processors
- SISD → classical processor
- SIMD → array processor
- MIMD → distributed or shared memory
- SPMD → single program & multiple data
- MPMD → multiple program & multiple data

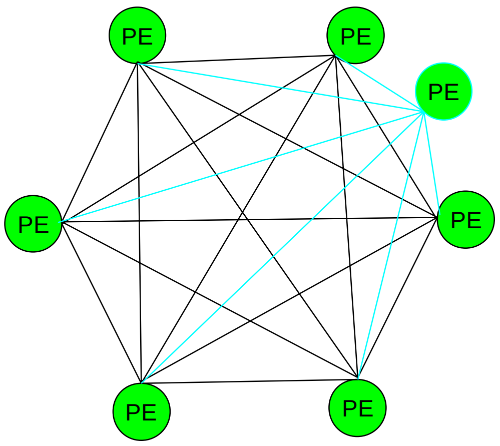


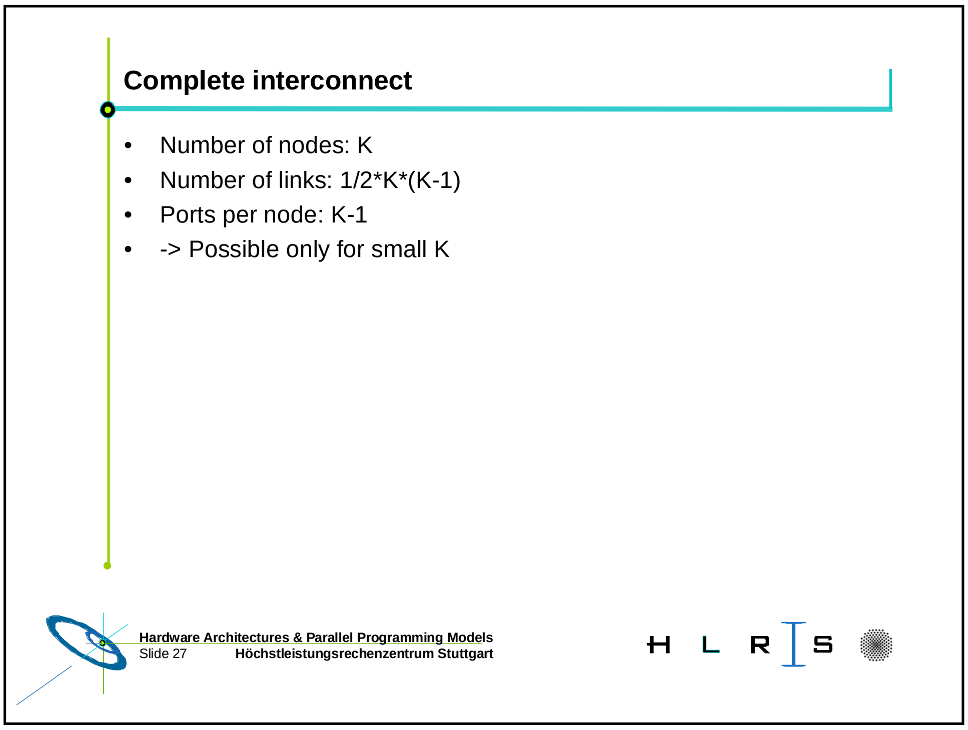
I heard that ....

- Network of workstations (NOW)? → Distributed memory
- Beowulf-class systems = Clusters of Commercial Off-The-Shelf (COTS) PCs? → Distributed memory
- Multiboard workstations/PCs ? → Shared memory
- SMP? → Symmetric multiprocessing.
  An easy way to do shared memory
- PVP? → Parallel vector processing. Merger of two good concepts
- MPP? → Massively parallel processing.

Network Topologies

Complete interconnect

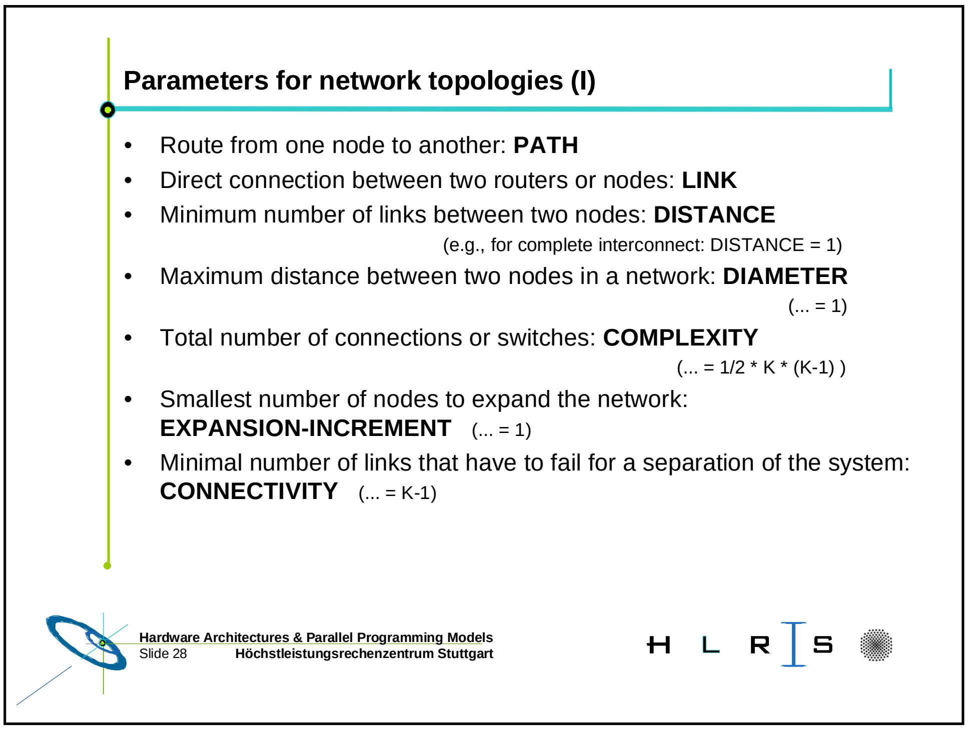




Complete interconnect

- Number of nodes: K
- Number of links: 1/2*K*(K-1)
- Ports per node: K-1
- -> Possible only for small K

Hardware Architectures & Parallel Programming Models
Slide 27 Höchstleistungsrechenzentrum Stuttgart



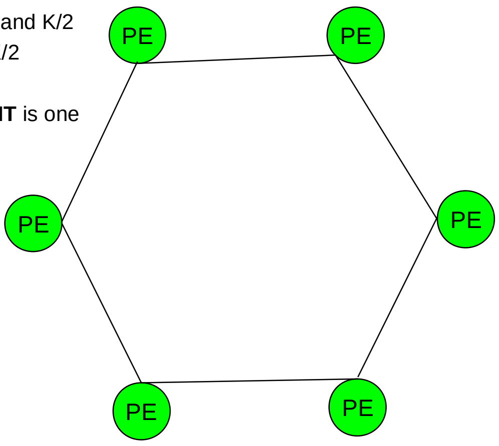
Parameters for network topologies (I)

- Route from one node to another: **PATH**
- Direct connection between two routers or nodes: **LINK**
- Minimum number of links between two nodes: **DISTANCE**
  (e.g., for complete interconnect: DISTANCE = 1)
- Maximum distance between two nodes in a network: **DIAMETER**
  (... = 1)
- Total number of connections or switches: **COMPLEXITY**
  (... = 1/2 * K * (K-1))
- Smallest number of nodes to expand the network:
  EXPANSION-INCREMENT (... = 1)
- Minimal number of links that have to fail for a separation of the system:
  CONNECTIVITY (... = K-1)

Hardware Architectures & Parallel Programming Models
Slide 28 Höchstleistungsrechenzentrum Stuttgart

Ring topology (Clusters)

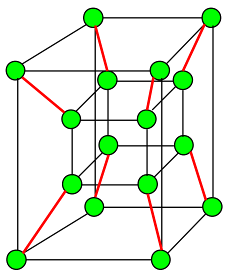
- **DISTANCE** between one and K/2
- **DIAMETER** is therefore K/2
- **COMPLEXITY** is K
- **EXPANSION-INCREMENT** is one
- **CONNECTIVITY** is 2



Hypercube (very popular in CS)

- **DISTANCE** between one and $\log_2(K)$
- **DIAMETER** is therefore $\log_2(K)$
- **COMPLEXITY** is $1/2*K*\log_2(K)$
- **EXPANSION-INCREMENT** is K
- **CONNECTIVITY** is $\log_2(K)$

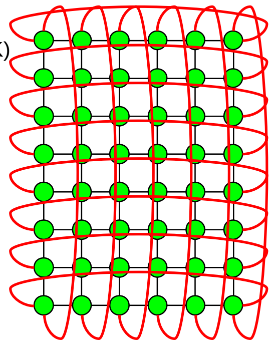
- Build in iPSC but still too complex and too expensive!!



2D-Mesh or Torus (Paragon, T3E)

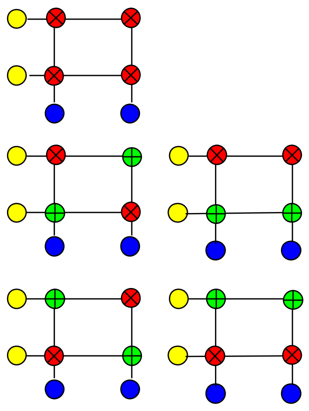
- **MESH**
- **DISTANCE** between one and ~2*sqrt(K)
- **DIAMETER** is therefore ~2*sqrt(K)
- **COMPLEXITY** is ~2*K
- **EXPANSION-INCREMENT** is ~sqrt(K)
- **CONNECTIVITY** is 2

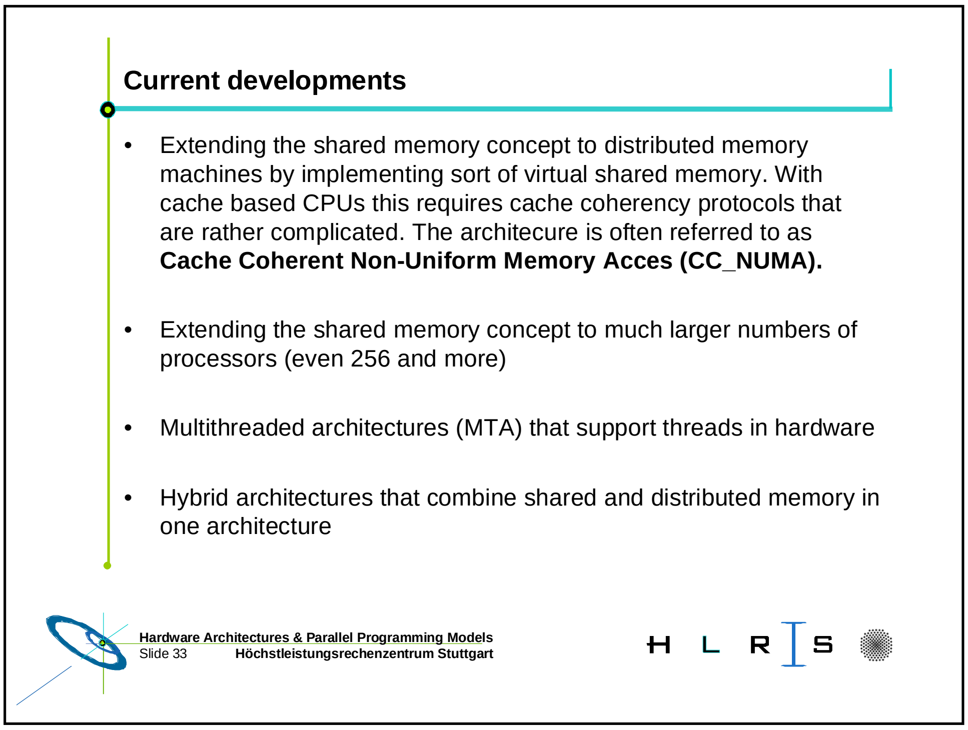
- **TORUS**
- **DISTANCE** between 1 and ~sqrt(K)
- **DIAMETER** is therefore ~sqrt(K)
- **COMPLEXITIY** is 2*K
- **EXPANSION INCREMENT** is ~sqrt(K)
- **CONNECTIVITY** is 4



Switch (SX-4, SX-5)

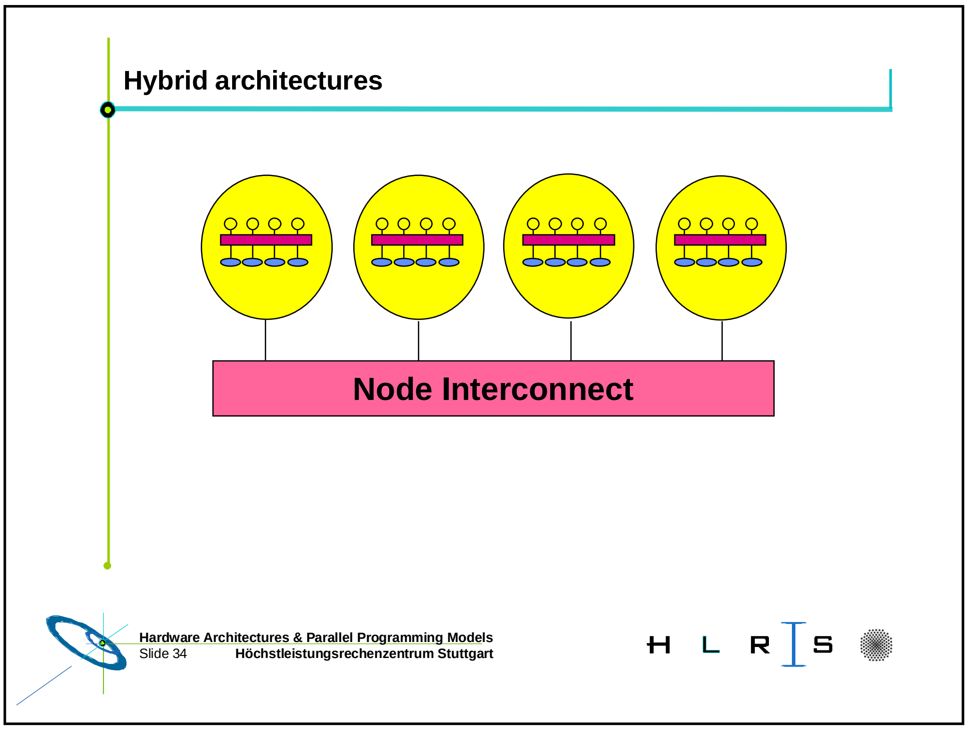
- Connects N CPUs to M memory banks
- Number of switching elements N*M



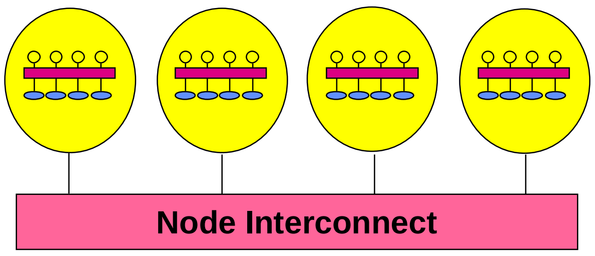


Current developments

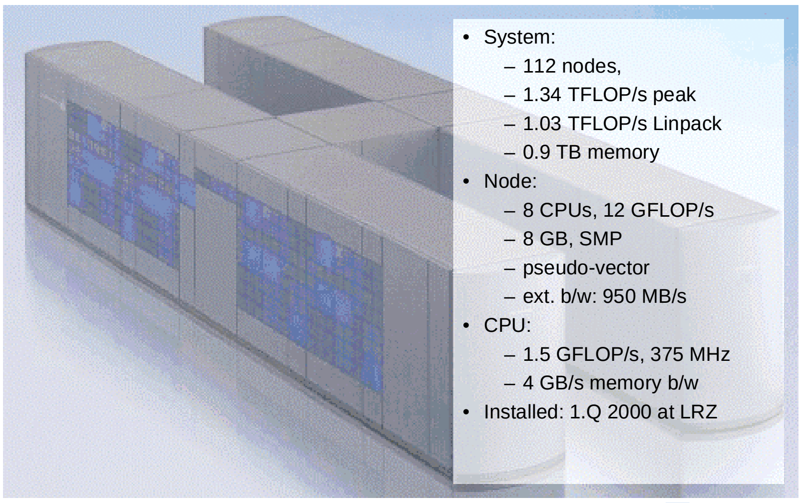
- Extending the shared memory concept to distributed memory machines by implementing sort of virtual shared memory. With cache based CPUs this requires cache coherency protocols that are rather complicated. The architecure is often referred to as **Cache Coherent Non-Uniform Memory Acces (CC_NUMA).**
- Extending the shared memory concept to much larger numbers of processors (even 256 and more)
- Multithreaded architectures (MTA) that support threads in hardware
- Hybrid architectures that combine shared and distributed memory in one architecture



Hybrid architectures

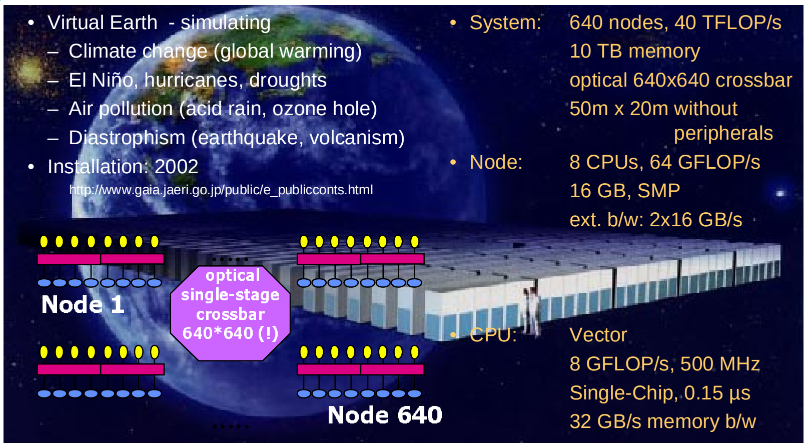


Hitachi SR 8000-F1/112 (Rank 5 in TOP 500 / June 2000)



- System:
  - 112 nodes,
  - 1.34 TFLOP/s peak
  - 1.03 TFLOP/s Linpack
  - 0.9 TB memory
- Node:
  - 8 CPUs, 12 GFLOP/s
  - 8 GB, SMP
  - pseudo-vector
  - ext. b/w: 950 MB/s
- CPU:
  - 1.5 GFLOP/s, 375 MHz
  - 4 GB/s memory b/w
- Installed: 1.Q 2000 at LRZ

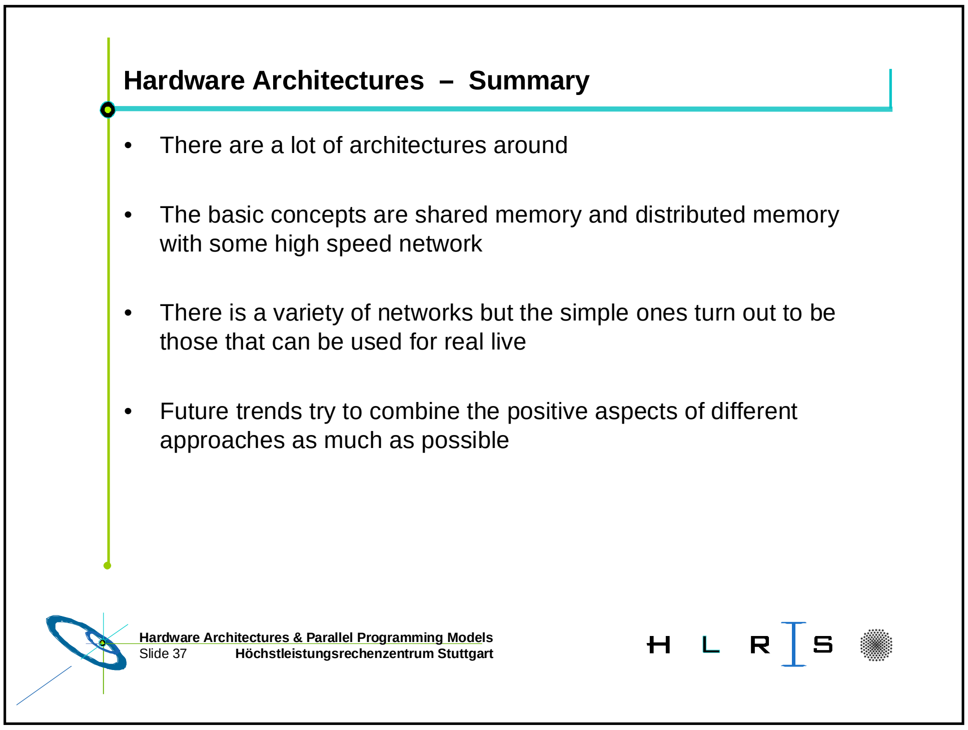
Earth Simulator Project ESRDC / GS 40 (NEC)



- Virtual Earth - simulating
  - Climate change (global warming)
  - El Niño, hurricanes, droughts
  - Air pollution (acid rain, ozone hole)
  - Diastrophism (earthquake, volcanism)
- Installation: 2002

  http://www.gaia.jaeri.go.jp/public/e_publicconts.html

• System:	640 nodes, 40 TFLOP/s 10 TB memory optical 640x640 crossbar 50m x 20m without peripherals
• Node:	8 CPUs, 64 GFLOP/s 16 GB, SMP ext. b/w: 2x16 GB/s
• CPU:	Vector 8 GFLOP/s, 500 MHz Single-Chip, 0.15 µs 32 GB/s memory b/w

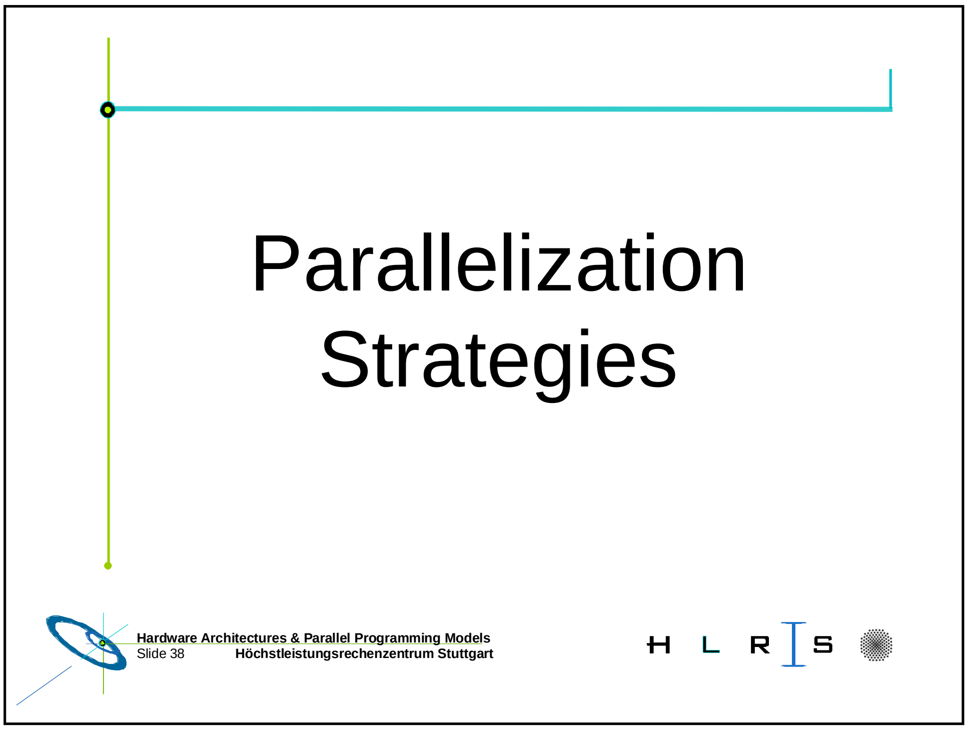


Hardware Architectures – Summary

- There are a lot of architectures around
- The basic concepts are shared memory and distributed memory with some high speed network
- There is a variety of networks but the simple ones turn out to be those that can be used for real live
- Future trends try to combine the positive aspects of different approaches as much as possible

Hardware Architectures & Parallel Programming Models
Slide 37 Höchstleistungsrechenzentrum Stuttgart

H L R S



Parallelization Strategies

Hardware Architectures & Parallel Programming Models
Slide 38 Höchstleistungsrechenzentrum Stuttgart

H L R S

Parallel Compiler

- Why can't I just say

  f90 -Parallel mycode.f

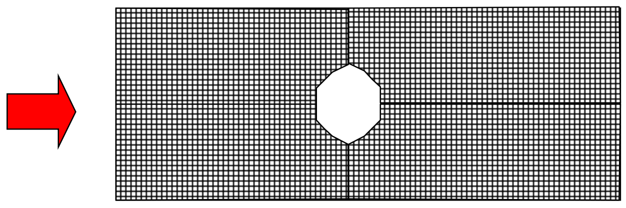
  or

  cc -Parallel mycode.c

  and everything works fine?

- Logical dependencies
- Data dependencies
- Global view

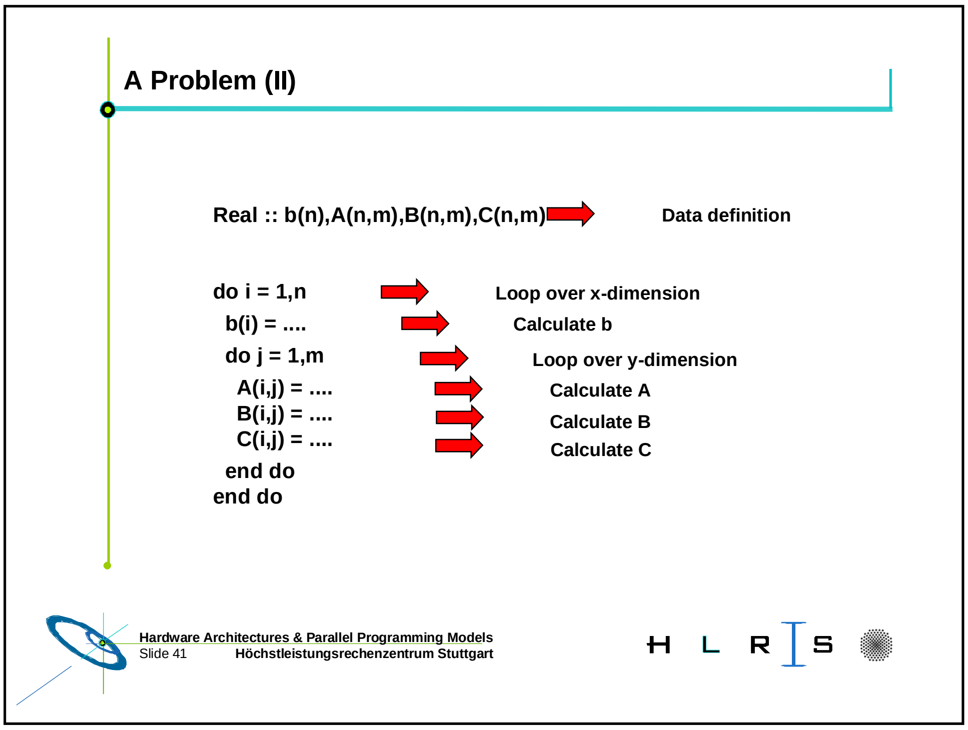
A Problem (I)



Flow around a cylinder:
Numerical Simulation using FV, FE or FD

Data Structure: A(1:n,1:m)

Solve: (A+B+C)x=b



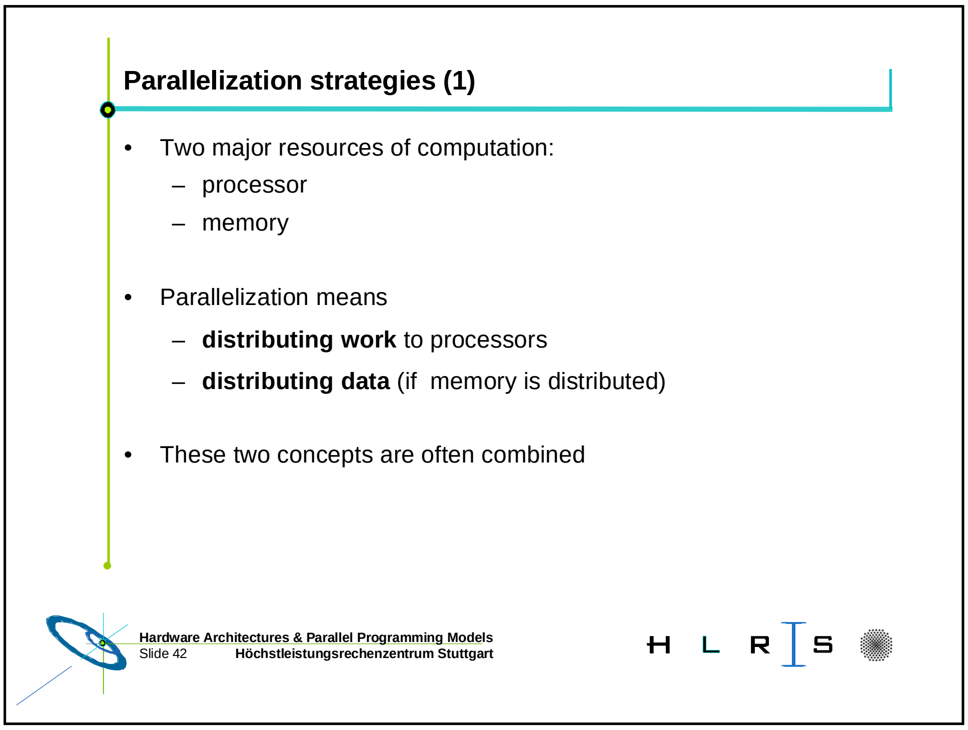
A Problem (II)

```
Real :: b(n),A(n,m),B(n,m),C(n,m)   ➡   Data definition

do i = 1,n          ➡   Loop over x-dimension
 b(i) = ....        ➡     Calculate b
 do j = 1,m         ➡       Loop over y-dimension
  A(i,j) = ....     ➡         Calculate A
  B(i,j) = ....     ➡         Calculate B
  C(i,j) = ....     ➡         Calculate C
 end do
end do
```

Hardware Architectures & Parallel Programming Models
Slide 41 Höchstleistungsrechenzentrum Stuttgart

H L R S



Parallelization strategies (1)

- Two major resources of computation:
  - processor
  - memory
- Parallelization means
  - **distributing work** to processors
  - **distributing data** (if memory is distributed)
- These two concepts are often combined

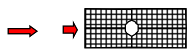
Hardware Architectures & Parallel Programming Models
Slide 42 Höchstleistungsrechenzentrum Stuttgart

H L R S

Parallelization strategies (2)

Work decomposition

(A+B+C)x = b
calc A
calc B
calc C
calc b



Flow around a cylinder:
Numerical Simulation using FV, FE or FD

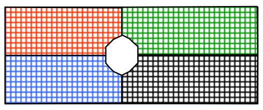
Data Structure: A(1:n,1:m)

Solve: (A+B+C)x=b

Data decomposition

A(1:20,1:50)
A(1:20,51:100)
A(1:20,101:150)
A(1:20,151:200)

Domain decomposition



Speedup, Efficency, and Scalup

- Definition:
  - T(p,N) = **time** to solve **problem of size N** on **p processors**
- Speedup:
  - S(p,N) = T(1,N) / T(p,N)
  - compute **same problem** with more processors in **shorter time**
- Efficiency:
  - E(p,N) = S(p,N) / p
- Scaleup:
  - Sc(p,N) = N / n   with T(1,n) = T(p,N)
  - compute **larger problem** with more processors in **same time**
- Problems:
  - Absolute MFLOPS rate **/** hardware peak performance?
  - S(p,N) close to **p** or far less? —> see Amdahls Law on next slide
  - Or super-scalar speedup: S(p,N)>p, e.g., due to cache usage

Parallelization problems

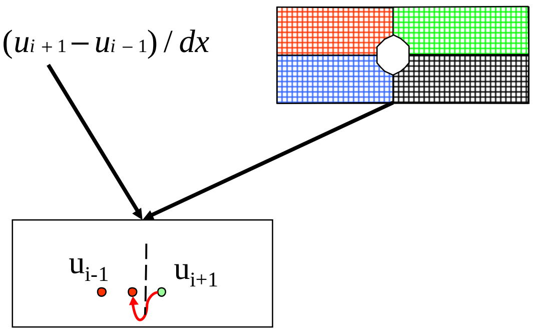
- Two major resources of computation:
  - processor
  - memory

- Parallelization means
  - distributing work to processors
    —> load balancing necessary
    —> synchronization overhead should be minimized
    —> to achieve optimal speedup
  - distributing data (if memory is distributed)
    —> implies communication to bring data to processor
    —> communication is overhead
    —> reduced speedup

Parallelization Problems

- Decomposition (Domain, Data, Work)
- Communication is overhead

$$du / dx = (u_{i+1} - u_{i-1}) / dx$$

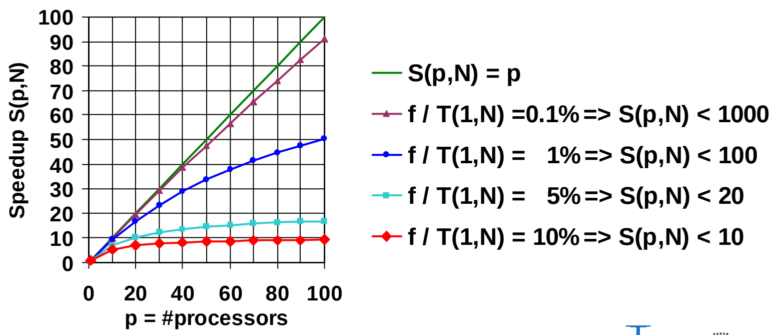


Amdahls Law

T(1,N) = f + (T(1,N) - f)   f ... sequential part of code that can not be done in parallel

S(p,N) = T(1,N) / T(p,N) = T(1,N) / **(**f + (T(1,N) - f) / p**)**

For p —> infinity, speedup is limited by S(p,N) < T(1,N) / f

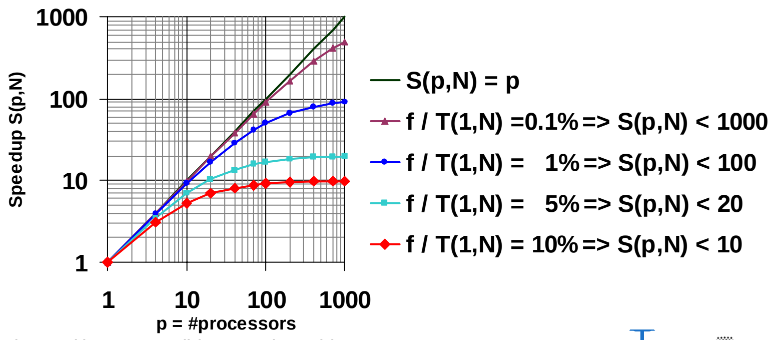


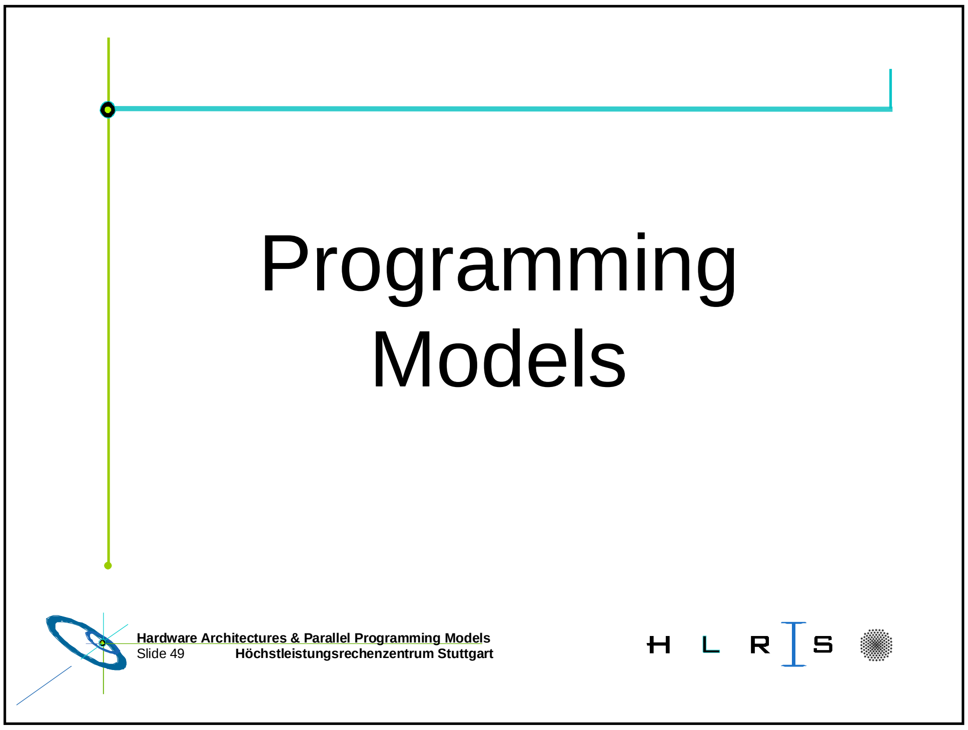
Amdahls Law (double-logarithmic)

T(1,N) = f + (T(1,N) - f)   f ... sequential part of code that can not be done in parallel

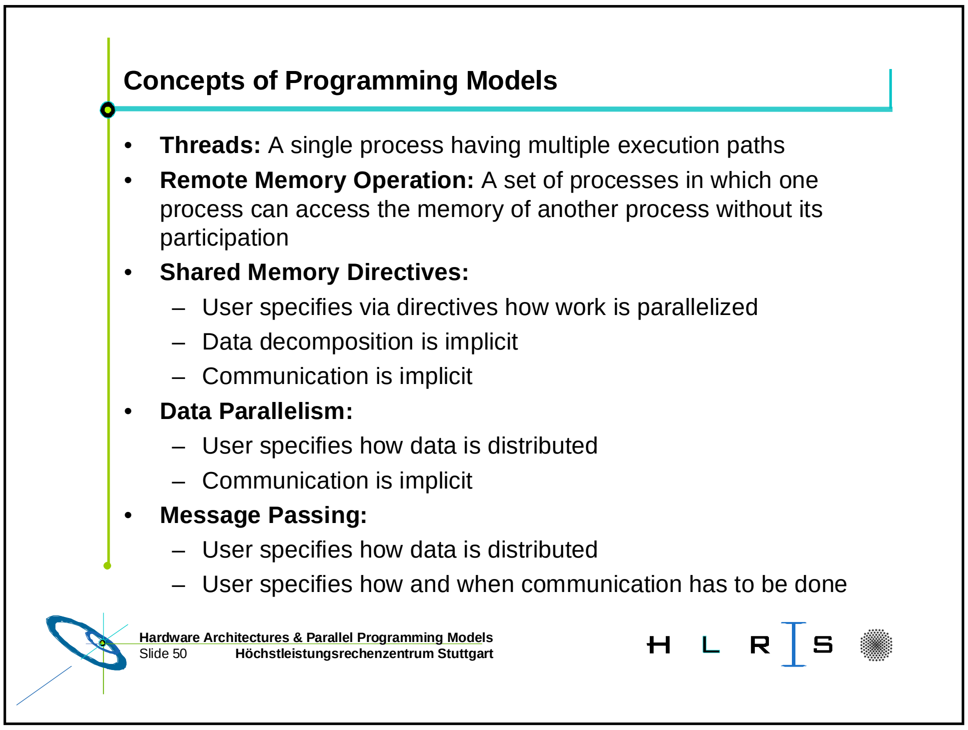
S(p,N) = T(1,N) / T(p,N) = T(1,N) / **(**f + (T(1,N) - f) / p**)**

For p —> infinity, speedup is limited by S(p,N) < T(1,N) / f



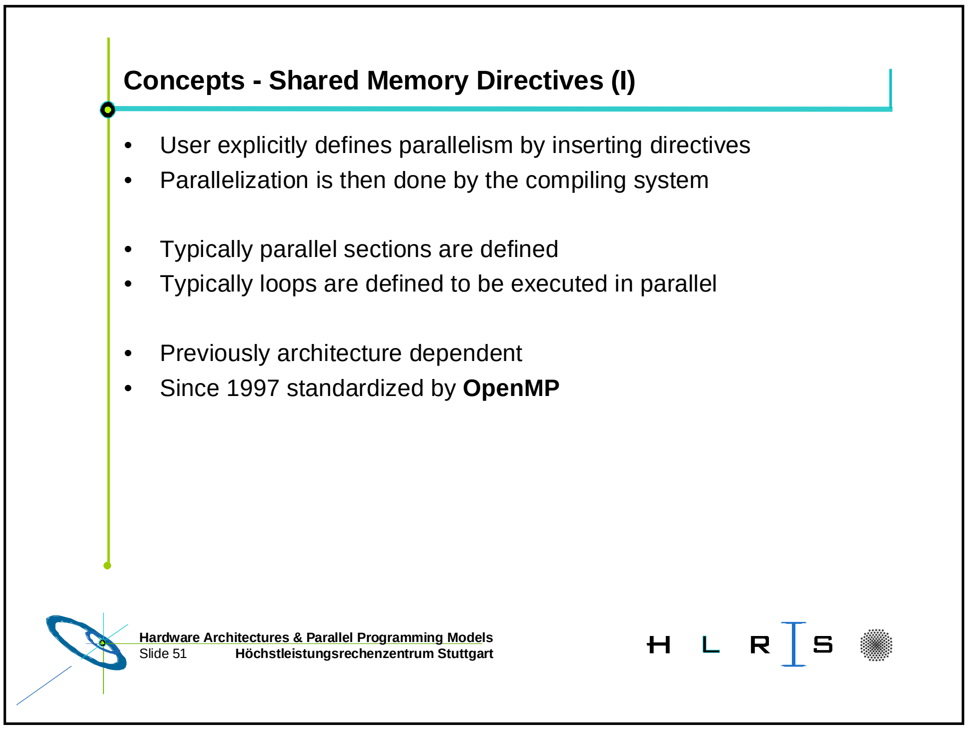


Programming Models



Concepts of Programming Models

- **Threads:** A single process having multiple execution paths
- **Remote Memory Operation:** A set of processes in which one process can access the memory of another process without its participation
- **Shared Memory Directives:**
  - User specifies via directives how work is parallelized
  - Data decomposition is implicit
  - Communication is implicit
- **Data Parallelism:**
  - User specifies how data is distributed
  - Communication is implicit
- **Message Passing:**
  - User specifies how data is distributed
  - User specifies how and when communication has to be done



Concepts - Shared Memory Directives (I)

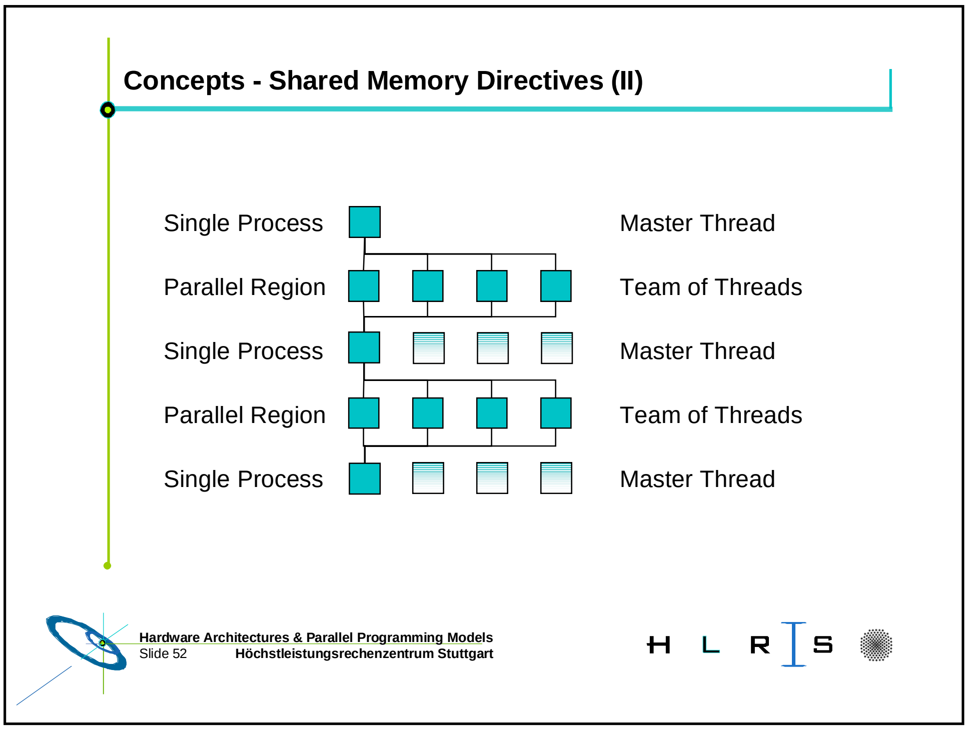
- User explicitly defines parallelism by inserting directives
- Parallelization is then done by the compiling system

- Typically parallel sections are defined
- Typically loops are defined to be executed in parallel

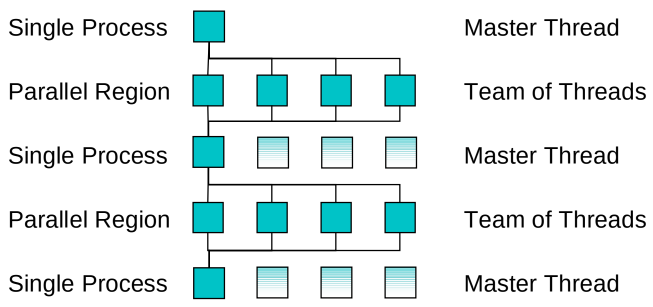
- Previously architecture dependent
- Since 1997 standardized by **OpenMP**

Hardware Architectures & Parallel Programming Models
Slide 51 Höchstleistungsrechenzentrum Stuttgart

H L R I S

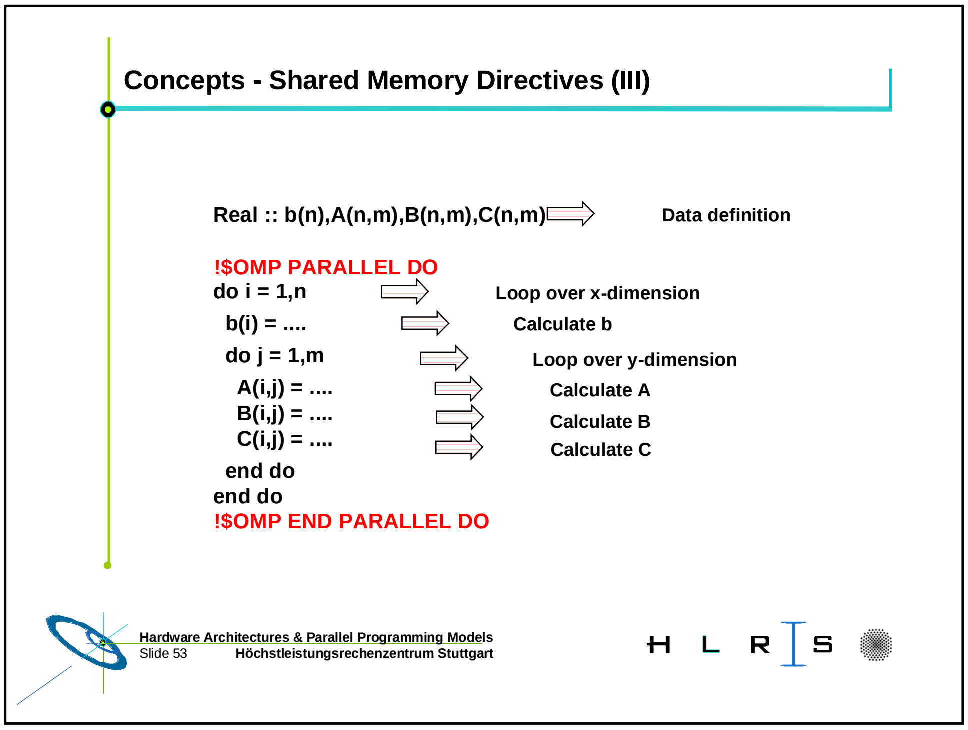


Concepts - Shared Memory Directives (II)



Hardware Architectures & Parallel Programming Models
Slide 52 Höchstleistungsrechenzentrum Stuttgart

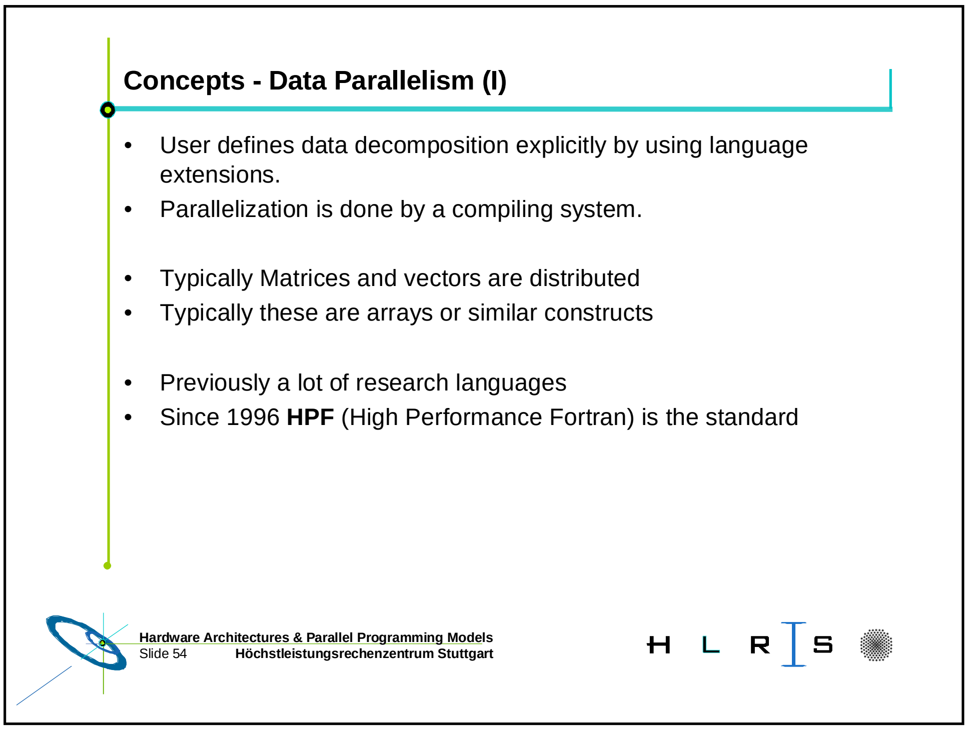
H L R I S



Concepts - Shared Memory Directives (III)

```
Real :: b(n),A(n,m),B(n,m),C(n,m) ⇨        Data definition

!$OMP PARALLEL DO
do i = 1,n            ⇨       Loop over x-dimension
  b(i) = ....          ⇨        Calculate b
  do j = 1,m            ⇨        Loop over y-dimension
    A(i,j) = ....        ⇨        Calculate A
    B(i,j) = ....        ⇨        Calculate B
    C(i,j) = ....        ⇨        Calculate C
  end do
end do
!$OMP END PARALLEL DO
```

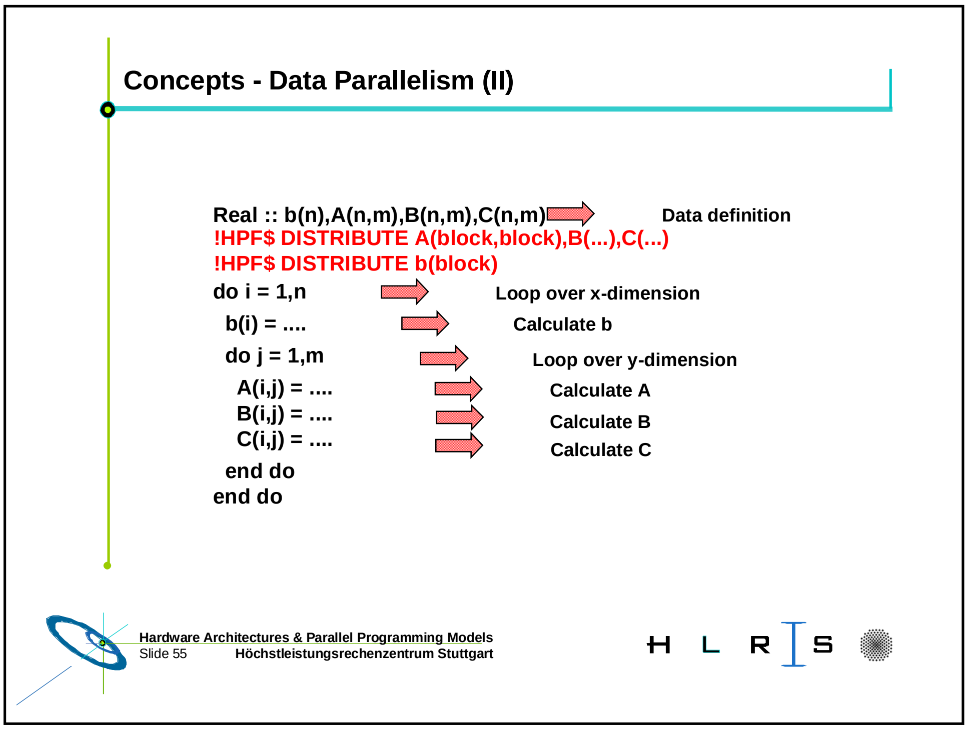


Concepts - Data Parallelism (I)

- User defines data decomposition explicitly by using language extensions.
- Parallelization is done by a compiling system.

- Typically Matrices and vectors are distributed
- Typically these are arrays or similar constructs

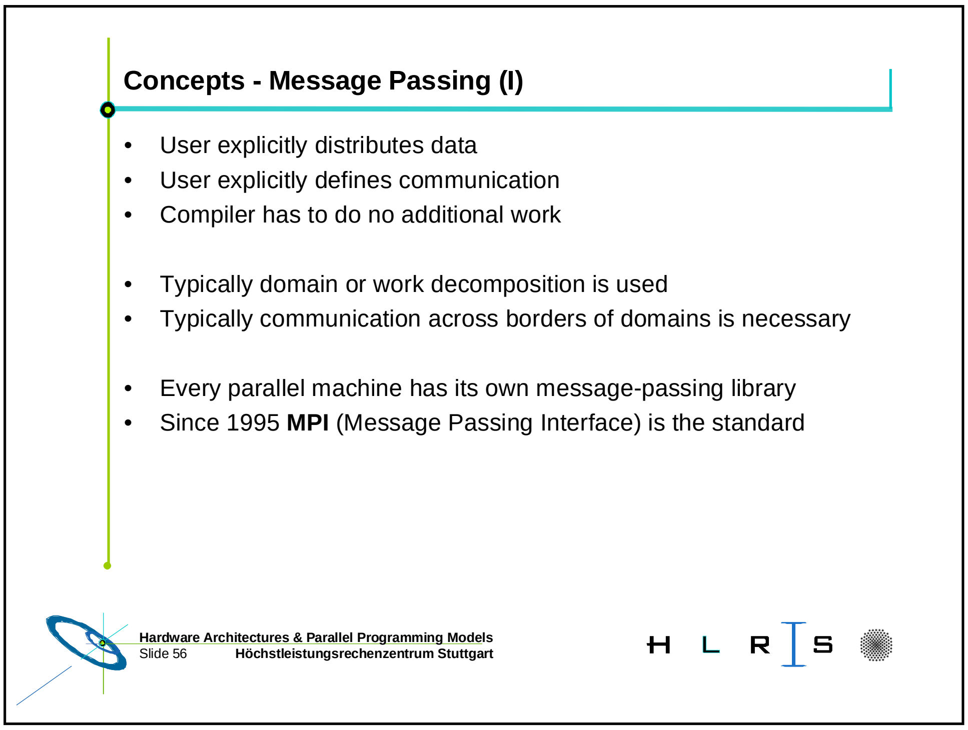
- Previously a lot of research languages
- Since 1996 **HPF** (High Performance Fortran) is the standard



Concepts - Data Parallelism (II)

```
Real :: b(n),A(n,m),B(n,m),C(n,m)          Data definition
!HPF$ DISTRIBUTE A(block,block),B(...),C(...)
!HPF$ DISTRIBUTE b(block)
do i = 1,n                            Loop over x-dimension
  b(i) = ....                           Calculate b
  do j = 1,m                              Loop over y-dimension
    A(i,j) = ....                           Calculate A
    B(i,j) = ....                           Calculate B
    C(i,j) = ....                           Calculate C
  end do
end do
```

Hardware Architectures & Parallel Programming Models
Slide 55 Höchstleistungsrechenzentrum Stuttgart



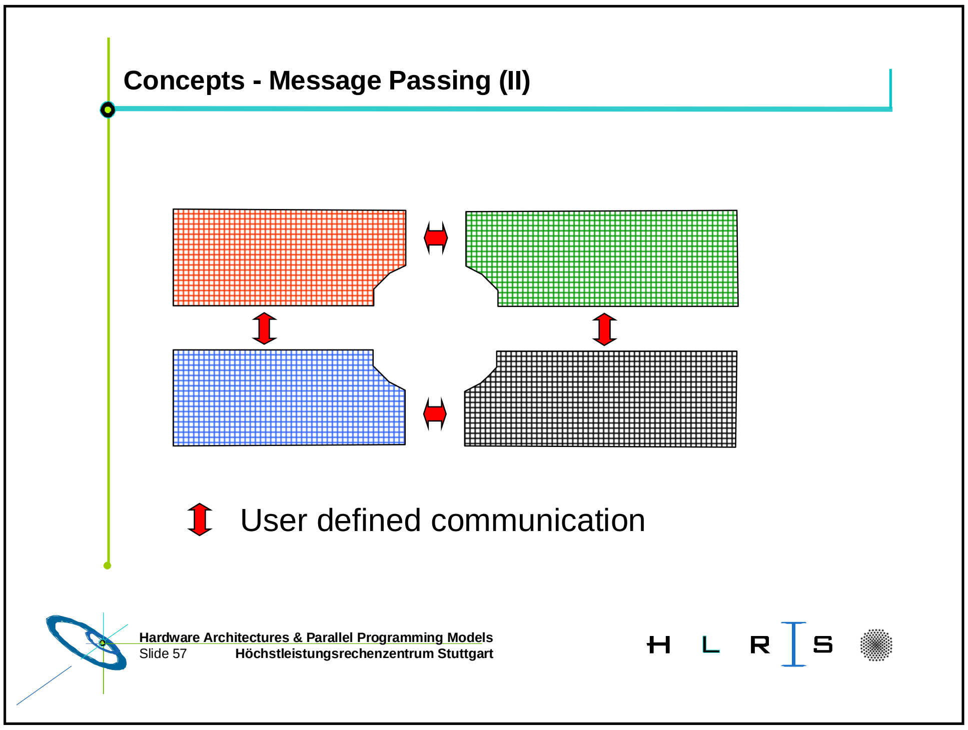
Concepts - Message Passing (I)

- User explicitly distributes data
- User explicitly defines communication
- Compiler has to do no additional work

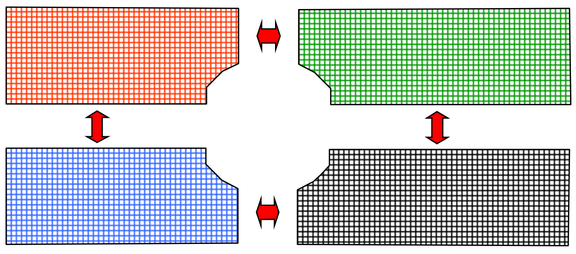
- Typically domain or work decomposition is used
- Typically communication across borders of domains is necessary

- Every parallel machine has its own message-passing library
- Since 1995 **MPI** (Message Passing Interface) is the standard

Hardware Architectures & Parallel Programming Models
Slide 56 Höchstleistungsrechenzentrum Stuttgart



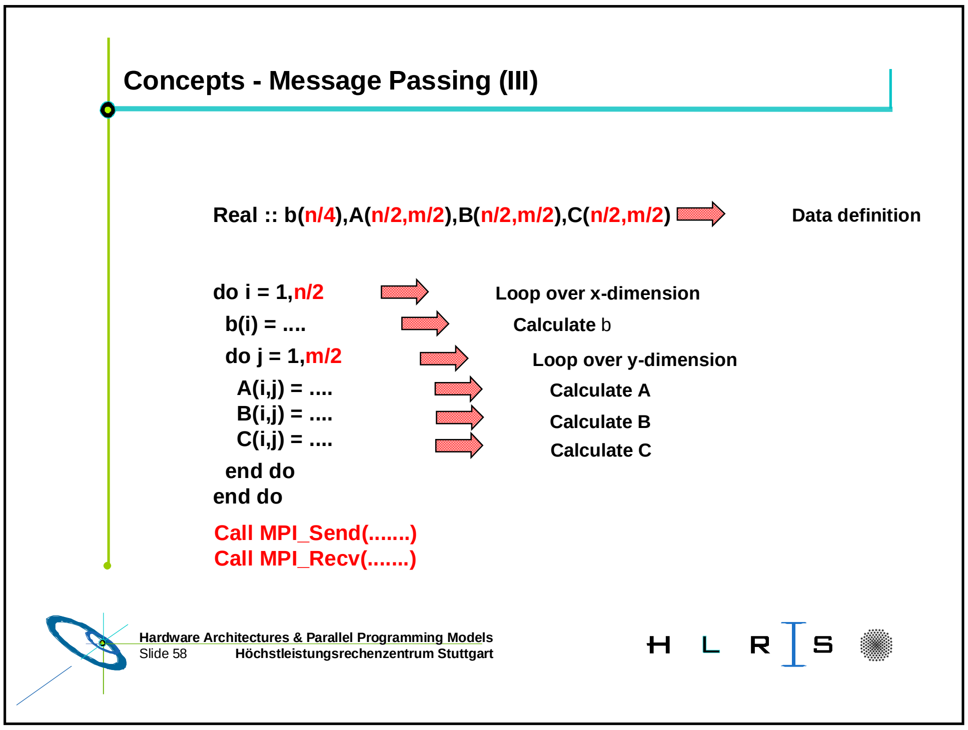
Concepts - Message Passing (II)



User defined communication

Hardware Architectures & Parallel Programming Models
Slide 57 Höchstleistungsrechenzentrum Stuttgart

H L R S



Concepts - Message Passing (III)

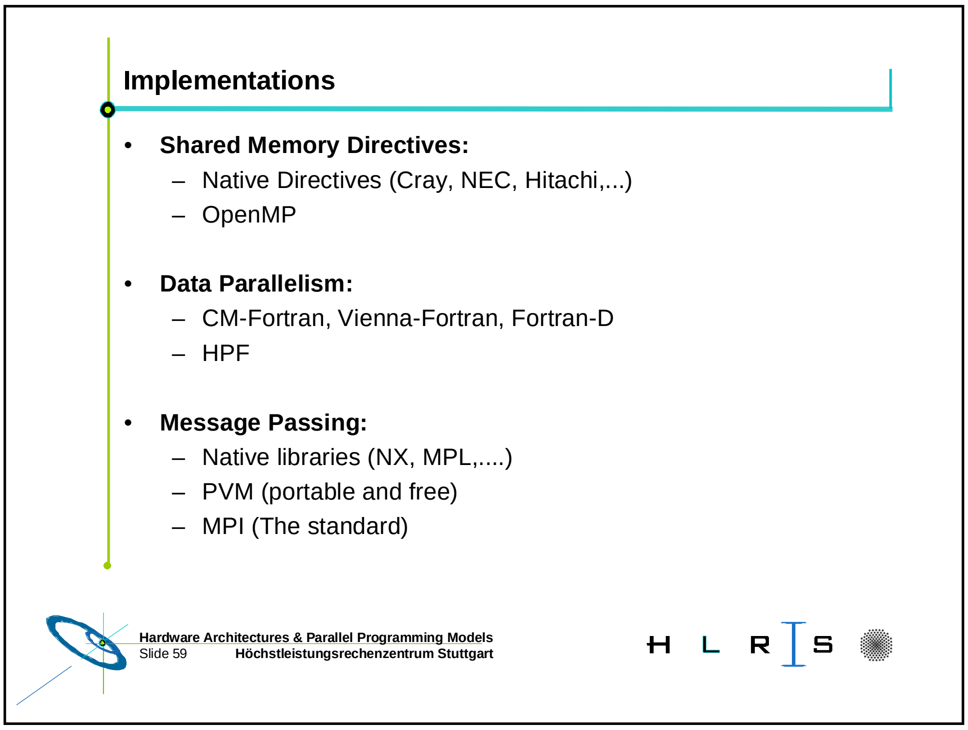
```
Real :: b(n/4),A(n/2,m/2),B(n/2,m/2),C(n/2,m/2)    ⇒  Data definition

do i = 1,n/2        ⇒  Loop over x-dimension
  b(i) = ....       ⇒    Calculate b
  do j = 1,m/2      ⇒      Loop over y-dimension
    A(i,j) = ....   ⇒        Calculate A
    B(i,j) = ....   ⇒        Calculate B
    C(i,j) = ....   ⇒        Calculate C
  end do
end do

Call MPI_Send(.......)
Call MPI_Recv(.......)
```

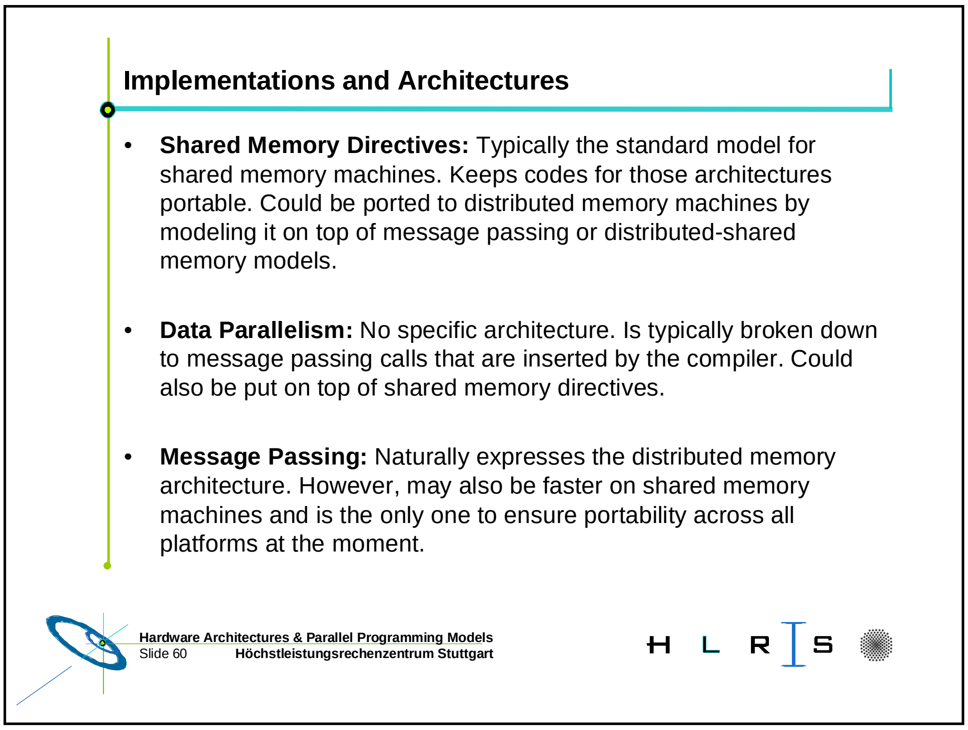
Hardware Architectures & Parallel Programming Models
Slide 58 Höchstleistungsrechenzentrum Stuttgart

H L R S



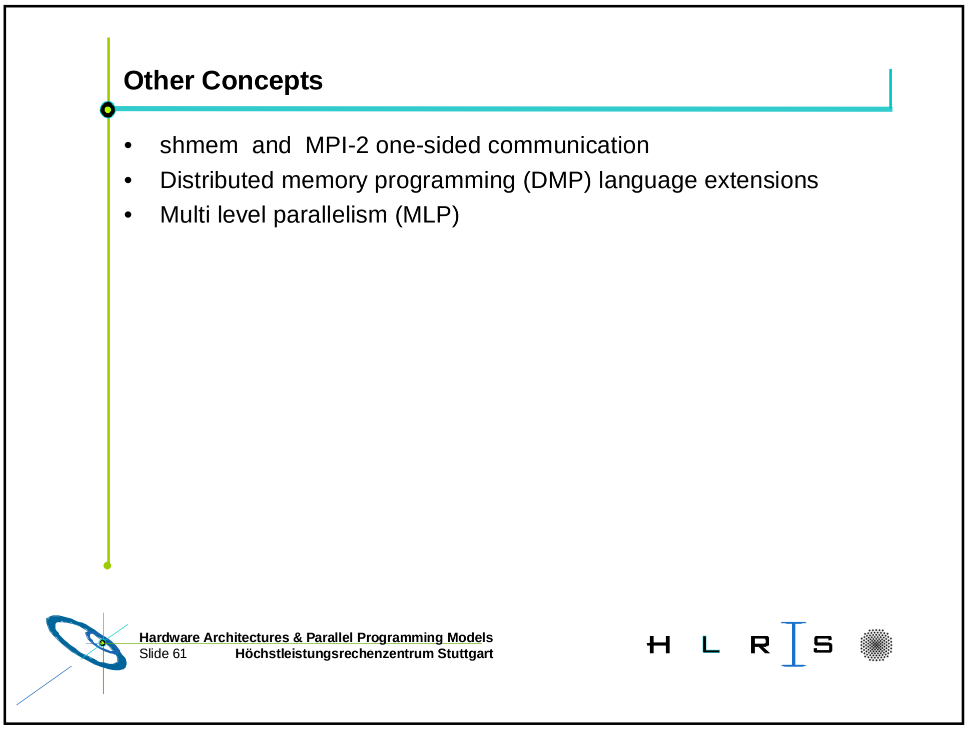
Implementations

- **Shared Memory Directives:**
  - Native Directives (Cray, NEC, Hitachi,...)
  - OpenMP
- **Data Parallelism:**
  - CM-Fortran, Vienna-Fortran, Fortran-D
  - HPF
- **Message Passing:**
  - Native libraries (NX, MPL,....)
  - PVM (portable and free)
  - MPI (The standard)

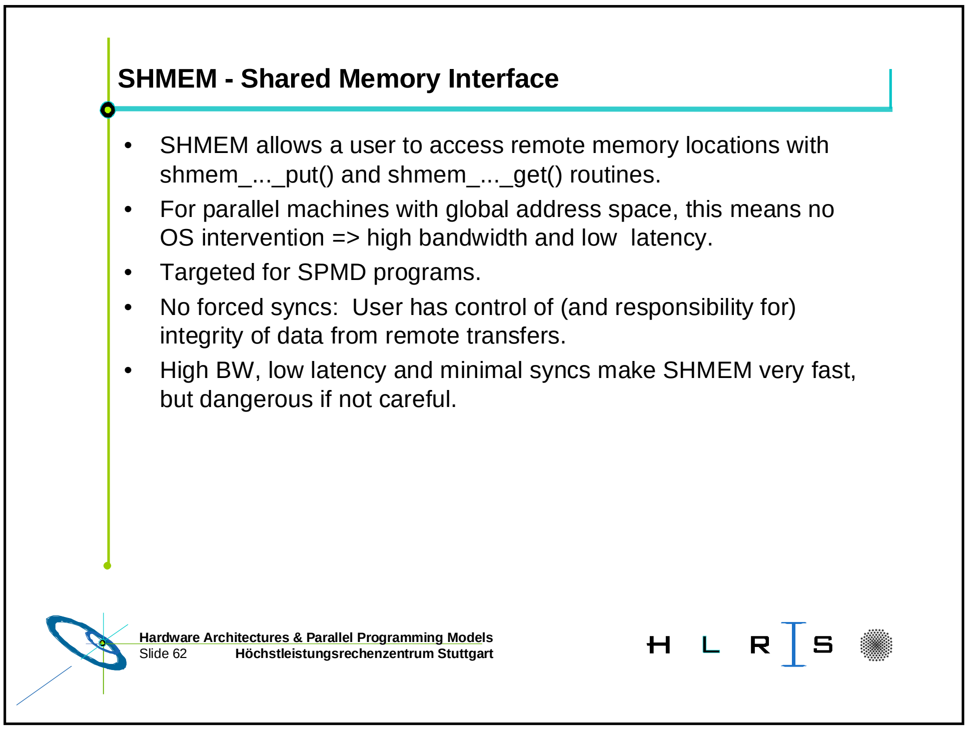


Implementations and Architectures

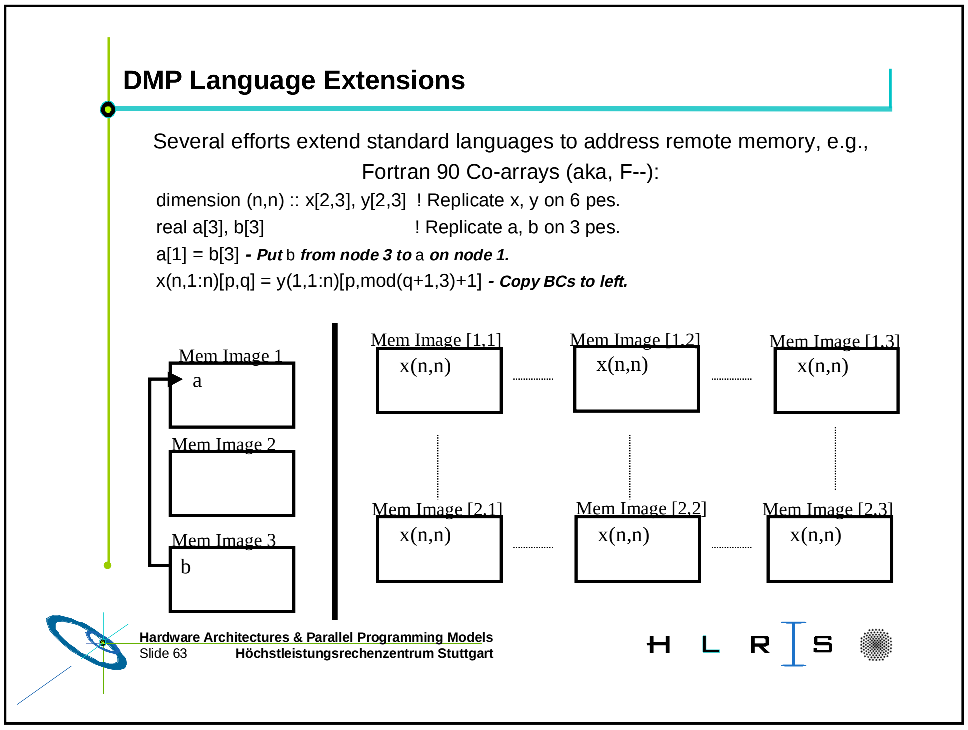
- **Shared Memory Directives:** Typically the standard model for shared memory machines. Keeps codes for those architectures portable. Could be ported to distributed memory machines by modeling it on top of message passing or distributed-shared memory models.
- **Data Parallelism:** No specific architecture. Is typically broken down to message passing calls that are inserted by the compiler. Could also be put on top of shared memory directives.
- **Message Passing:** Naturally expresses the distributed memory architecture. However, may also be faster on shared memory machines and is the only one to ensure portability across all platforms at the moment.


Other Concepts

- shmem and MPI-2 one-sided communication
- Distributed memory programming (DMP) language extensions
- Multi level parallelism (MLP)


SHMEM - Shared Memory Interface

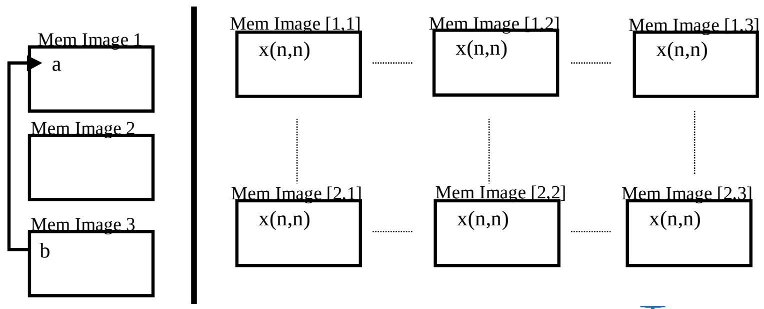
- SHMEM allows a user to access remote memory locations with shmem_..._put() and shmem_..._get() routines.
- For parallel machines with global address space, this means no OS intervention => high bandwidth and low latency.
- Targeted for SPMD programs.
- No forced syncs: User has control of (and responsibility for) integrity of data from remote transfers.
- High BW, low latency and minimal syncs make SHMEM very fast, but dangerous if not careful.



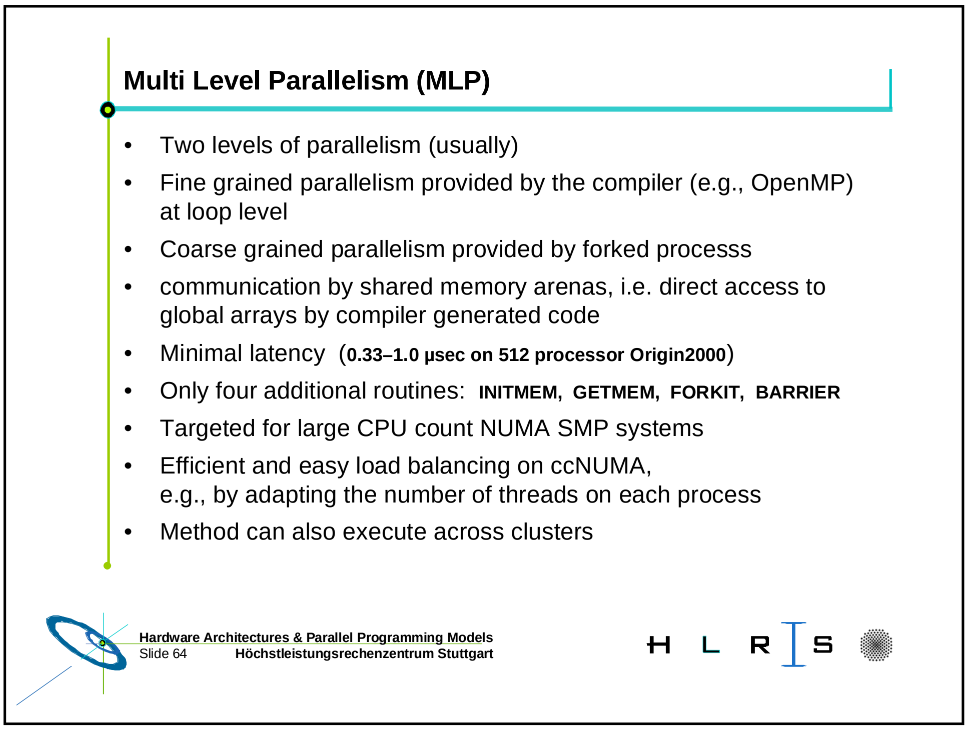
DMP Language Extensions

Several efforts extend standard languages to address remote memory, e.g.,
Fortran 90 Co-arrays (aka, F--):

dimension (n,n) :: x[2,3], y[2,3] ! Replicate x, y on 6 pes.
real a[3], b[3] ! Replicate a, b on 3 pes.
a[1] = b[3] ***- Put*** b ***from node 3 to*** a ***on node 1.***
x(n,1:n)[p,q] = y(1,1:n)[p,mod(q+1,3)+1] ***- Copy BCs to left.***



Hardware Architectures & Parallel Programming Models
Slide 63 Höchstleistungsrechenzentrum Stuttgart

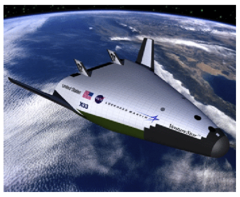


Multi Level Parallelism (MLP)

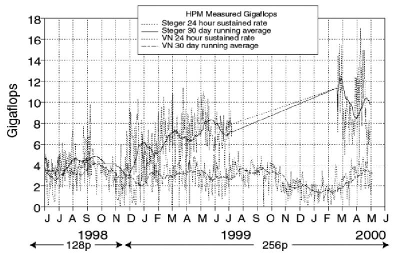
- Two levels of parallelism (usually)
- Fine grained parallelism provided by the compiler (e.g., OpenMP) at loop level
- Coarse grained parallelism provided by forked processs
- communication by shared memory arenas, i.e. direct access to global arrays by compiler generated code
- Minimal latency (**0.33–1.0 μsec on 512 processor Origin2000**)
- Only four additional routines: **INITMEM, GETMEM, FORKIT, BARRIER**
- Targeted for large CPU count NUMA SMP systems
- Efficient and easy load balancing on ccNUMA, e.g., by adapting the number of threads on each process
- Method can also execute across clusters

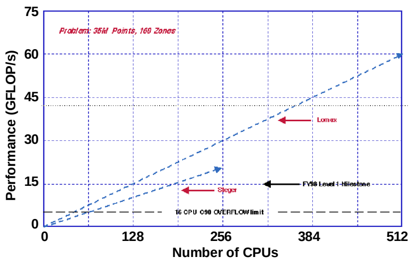
Hardware Architectures & Parallel Programming Models
Slide 64 Höchstleistungsrechenzentrum Stuttgart

Example: Parallel Efficiency of OVERFLOW/MLP



- OVERFLOW CFD code at NASA/Ames
- high, sustained GFLOP/s rate
- with Multi Level Parallelism (MLP)
- scalable on large CPU counts
- on 512 processor ccNUMA Origin 2000





MLPlib

The MLPlib routines for scalable parallel execution support are:

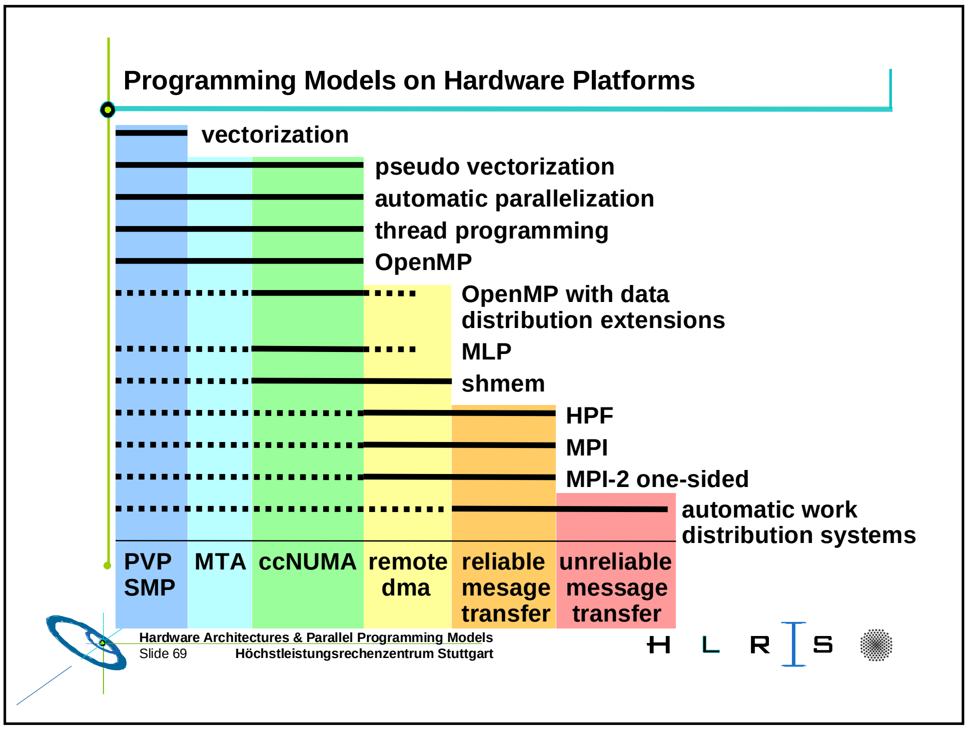
- Subroutine INITMEM(numbytes)
  - The INITMEM routine sets up a UNIX shared memory arena consisting of numbytes bytes to be used by all subsequently spawned processes
- Subroutine GETMEM(xarry,xpoint,numxbyt)
  - The GETMEM routine allocates numxbyt bytes to the xarray variable
  - xpoint is the Cray pointer to xarray
  - xarray is resident in the shared memory arena
  - The xarray data will be visible to all MLP processes using the shared memory arena .
- Subroutine FORKIT(numpro,myrank)
  - spawns a total of numpro additional processes
  - returns current process id myrank (0–numpro)
- Subroutine BARRIER(numpro)
  - The BARRIER roitine waits until numpro processes have hit the barrier, then all drop through

Reference: Ciotti, Taft, Peterson: “Early Experiences with the 512p Origin2000” in proceedings of the Cray User Group conference SUMMIT 2000, www.cray.org

Advantages and Challenges

Advantages and Challenges

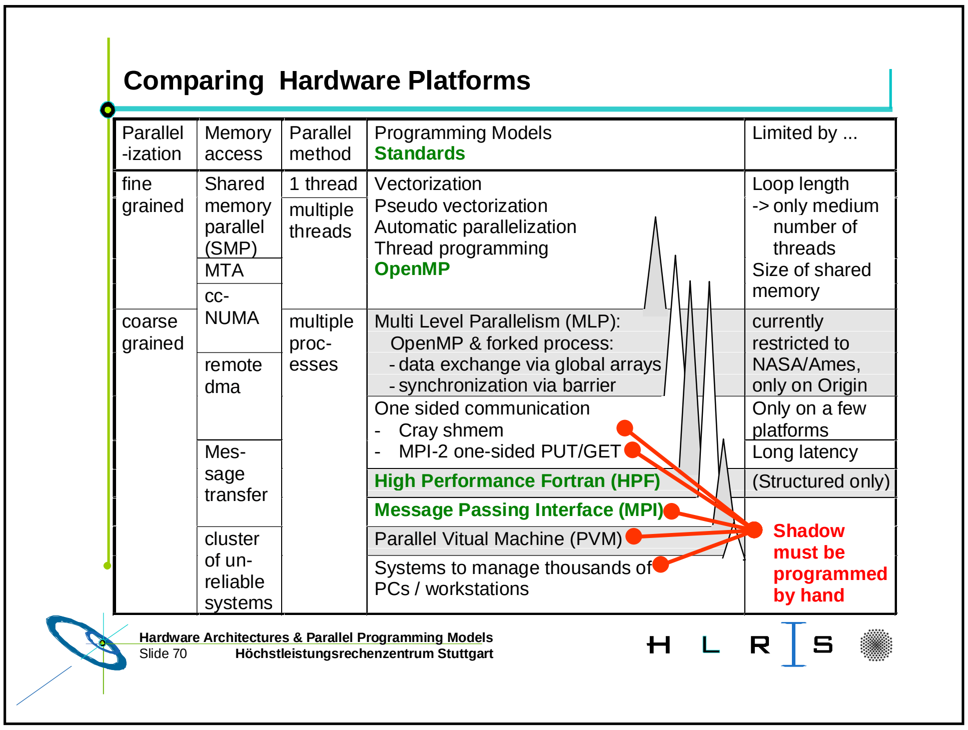
	OpenMP	HPF	MPI
Maturity of programming model	++	+	++
Maturity of standardization	+	+	++
Migration of serial programs	++	0	– –
Ease of programming (new progr.)	++	+	–
Correctness of parallelization	–	++	– –
Portability to any hardware architecture	–	++	++
Availability of implementations of the stand.	+	+	++
Availability of parallel libraries	0	0	0
Scalability to hundreds/thousands of processors	– –	0	++
Efficiency	–	0	++
Flexibility – dynamic program structures	–	–	++
– irregular grids, triangles, tetra-hedrons, load balancing, redistribut.	–	–	++



Programming Models on Hardware Platforms

Hardware Architectures & Parallel Programming Models
Slide 69 Höchstleistungsrechenzentrum Stuttgart

H L R S

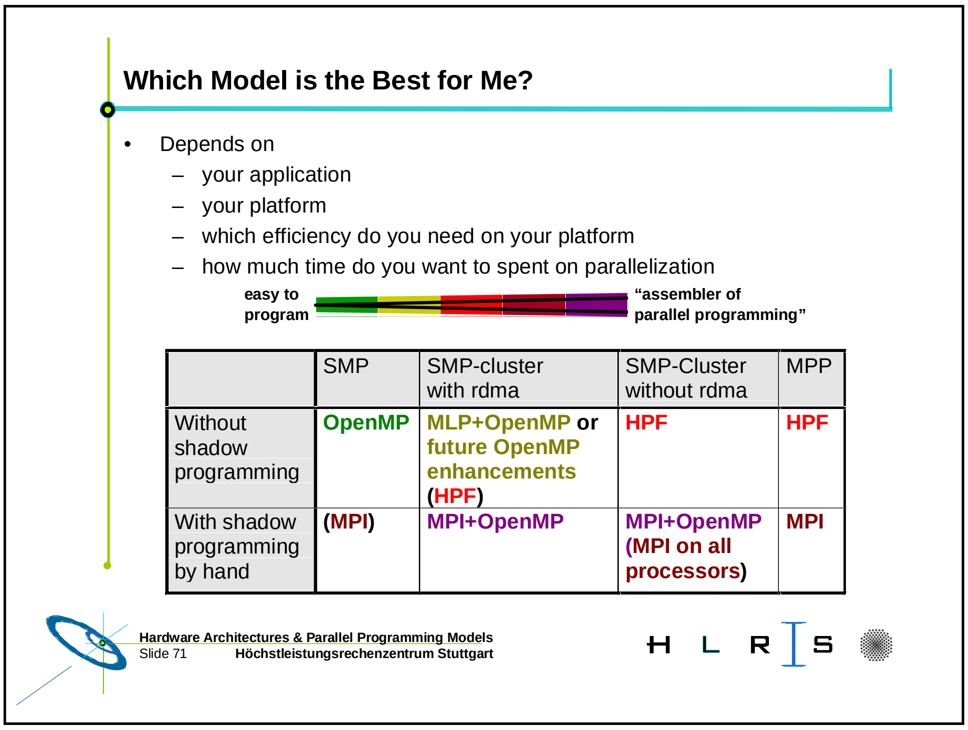


Comparing Hardware Platforms

Parallel -ization	Memory access	Parallel method	Programming Models **Standards**	Limited by ...
fine grained	Shared memory parallel (SMP)	1 thread	Vectorization	Loop length -> only medium number of threads
	MTA	multiple threads	Pseudo vectorization Automatic parallelization Thread programming **OpenMP**	Size of shared memory
	cc-NUMA			
coarse grained		multiple proc-esses	Multi Level Parallelism (MLP): OpenMP & forked process: - data exchange via global arrays - synchronization via barrier	currently restricted to NASA/Ames, only on Origin
	remote dma		One sided communication - Cray shmem - MPI-2 one-sided PUT/GET	Only on a few platforms Long latency
	Mes-sage transfer		**High Performance Fortran (HPF)**	(Structured only)
			Message Passing Interface (MPI)	**Shadow must be programmed by hand**
			Parallel Vitual Machine (PVM)	
	cluster of un-reliable systems		Systems to manage thousands of PCs / workstations	

Hardware Architectures & Parallel Programming Models
Slide 70 Höchstleistungsrechenzentrum Stuttgart

H L R S



Which Model is the Best for Me?

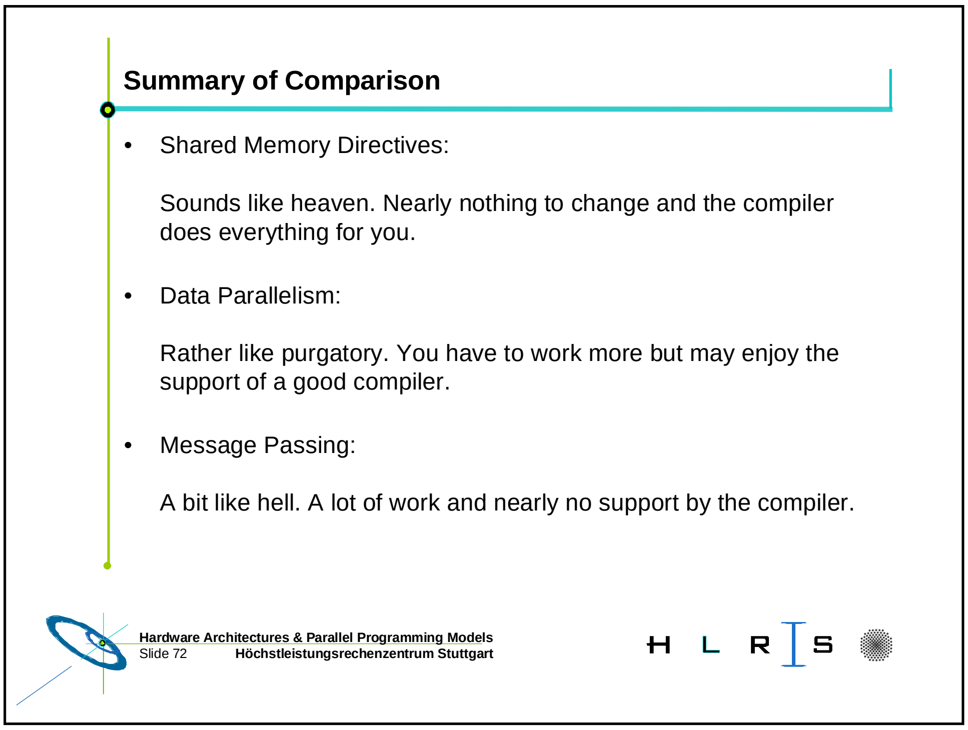
- Depends on
  - your application
  - your platform
  - which efficiency do you need on your platform
  - how much time do you want to spent on parallelization

easy to program ⟷ **“assembler of parallel programming”**

	SMP	SMP-cluster with rdma	SMP-Cluster without rdma	MPP
Without shadow programming	**OpenMP**	**MLP+OpenMP or future OpenMP enhancements (HPF)**	**HPF**	**HPF**
With shadow programming by hand	**(MPI)**	**MPI+OpenMP**	**MPI+OpenMP (MPI on all processors)**	**MPI**

Hardware Architectures & Parallel Programming Models
Slide 71 Höchstleistungsrechenzentrum Stuttgart

H L R I S



Summary of Comparison

- Shared Memory Directives:

  Sounds like heaven. Nearly nothing to change and the compiler does everything for you.

- Data Parallelism:

  Rather like purgatory. You have to work more but may enjoy the support of a good compiler.

- Message Passing:

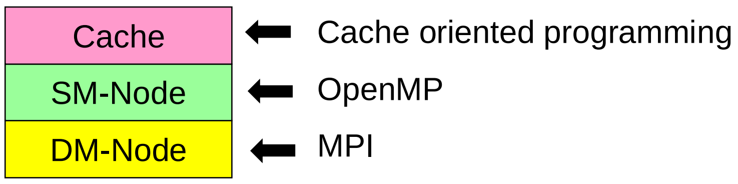
  A bit like hell. A lot of work and nearly no support by the compiler.

Hardware Architectures & Parallel Programming Models
Slide 72 Höchstleistungsrechenzentrum Stuttgart

H L R I S

Future Directions

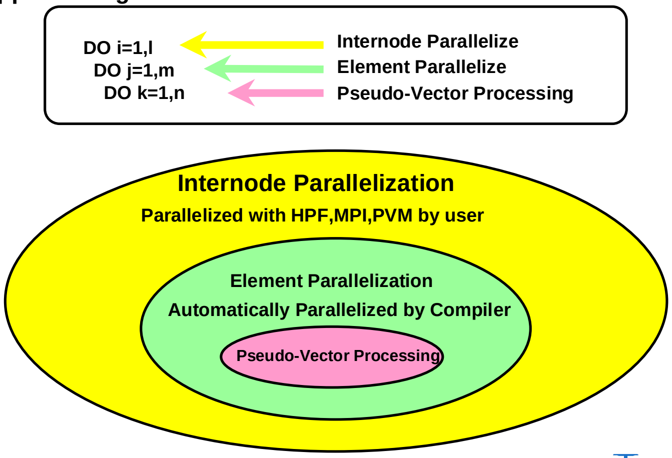
- Hierarchical Programming Models



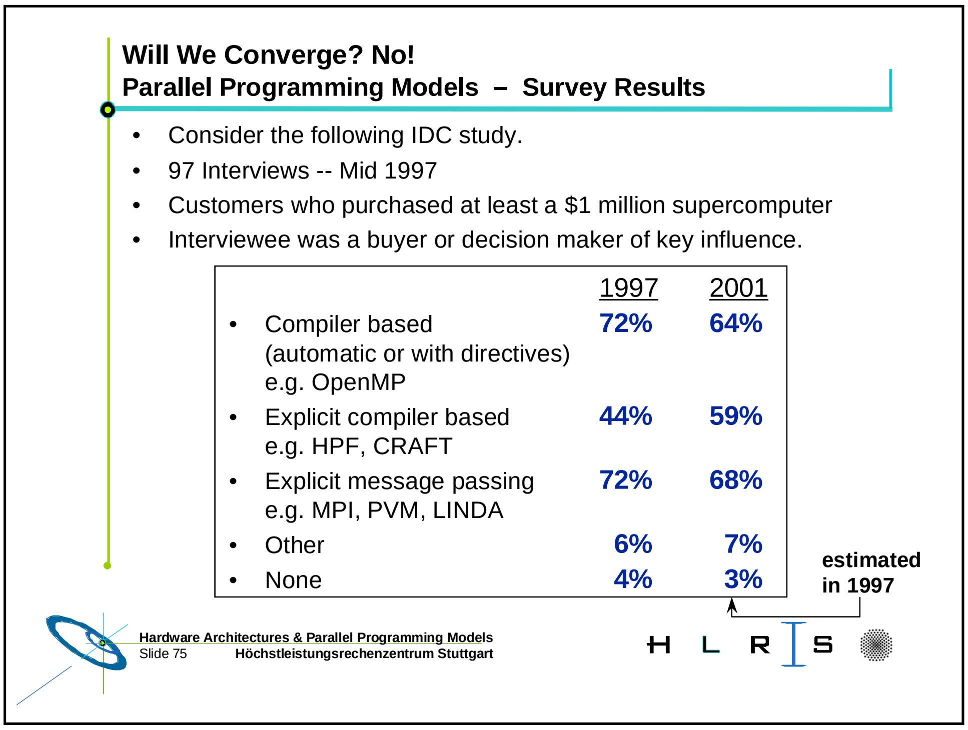
Hardware Architectures & Parallel Programming Models
Slide 73 Höchstleistungsrechenzentrum Stuttgart

Application Program Processing

Applied Image



Hardware Architectures & Parallel Programming Models
Slide 74 Höchstleistungsrechenzentrum Stuttgart



Will We Converge? No!
Parallel Programming Models – Survey Results

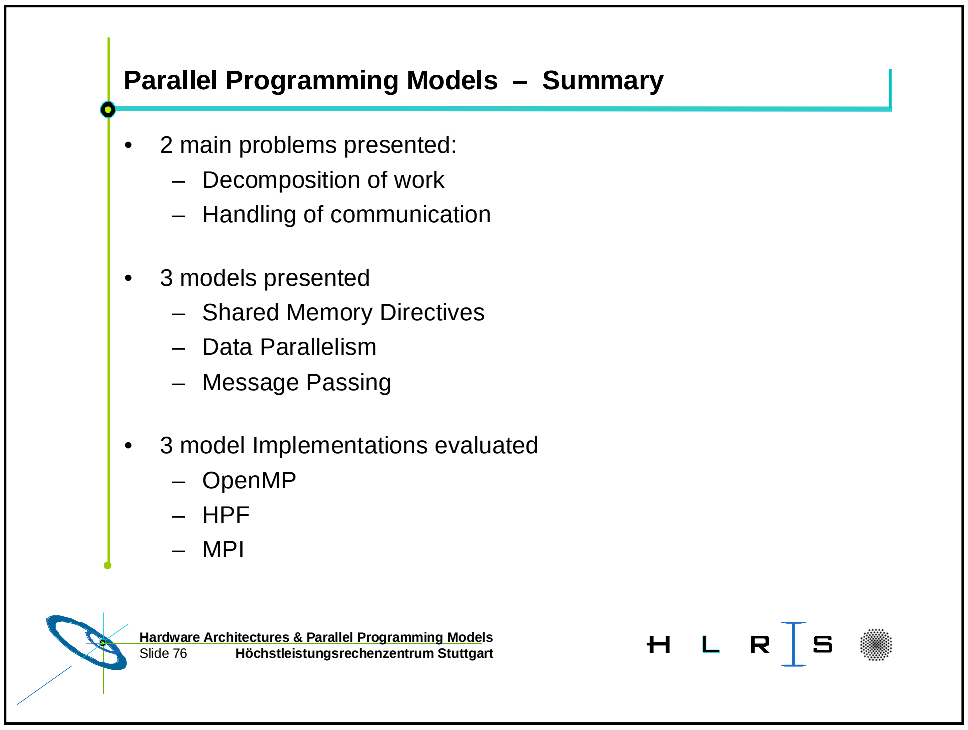
- Consider the following IDC study.
- 97 Interviews -- Mid 1997
- Customers who purchased at least a $1 million supercomputer
- Interviewee was a buyer or decision maker of key influence.

	1997	2001
• Compiler based (automatic or with directives) e.g. OpenMP	**72%**	**64%**
• Explicit compiler based e.g. HPF, CRAFT	**44%**	**59%**
• Explicit message passing e.g. MPI, PVM, LINDA	**72%**	**68%**
• Other	**6%**	**7%**
• None	**4%**	**3%**

estimated in 1997

Hardware Architectures & Parallel Programming Models
Slide 75 Höchstleistungsrechenzentrum Stuttgart

H L R S



Parallel Programming Models – Summary

- 2 main problems presented:
  - Decomposition of work
  - Handling of communication
- 3 models presented
  - Shared Memory Directives
  - Data Parallelism
  - Message Passing
- 3 model Implementations evaluated
  - OpenMP
  - HPF
  - MPI

Hardware Architectures & Parallel Programming Models
Slide 76 Höchstleistungsrechenzentrum Stuttgart

H L R S