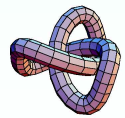
Chemnitz High Performance Linux Cluster
– First Experiences on CHiC –

Matthias Pester
pester@mathematik.tu-chemnitz.de

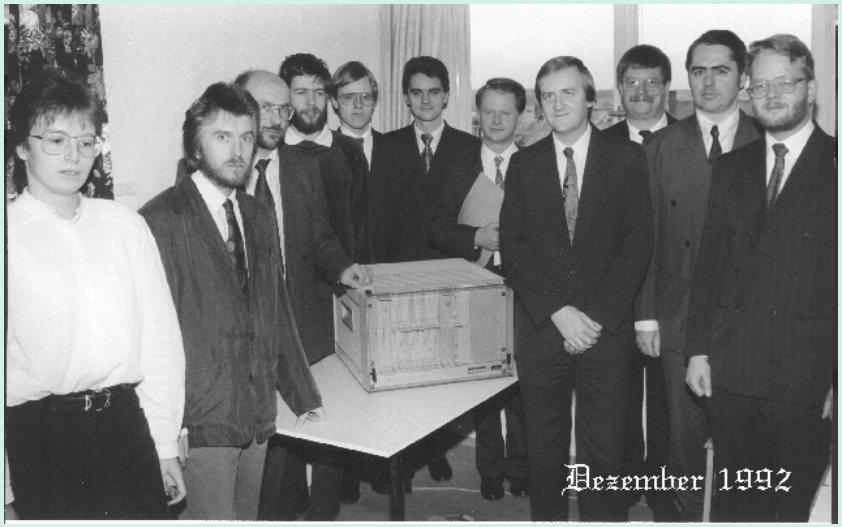
Fakultät für Mathematik
Technische Universität Chemnitz

Symposium Wissenschaftlich-technisches Hochleistungsrechnen
23. März 2007



Outline

Milestones of Chemnitz Parallel Computers



Start-up:	1992
Multicluster	32× T800-20
CLiC	528× PIII-800 MHz,
GC/PP	128× PPC 601-80,
Multicluster	32× T800-20,

Milestones of Chemnitz Parallel Computers

Start-up:	1994
GC/PP	128× PPC 601-80
CLiC	528× PIII-800 MHz,
GC/PP	128× PPC 601-80,
1992: Multicluster	32× T800-20,

Milestones of Chemnitz Parallel Computers

Start-up:	2000
CLiC	528× PIII-800 MHz
CLiC	528× PIII-800 MHz,
1994: GC/PP	128× PPC 601-80,
1992: Multicluster	32× T800-20,

Milestones of Chemnitz Parallel Computers

Start-up:	2007
2007: CHiC	538× Opteron 4×2,6 GHz
2000: CLiC	528× PIII-800 MHz,
1994: GC/PP	128× PPC 601-80,
1992: Multicluster	32× T800-20,

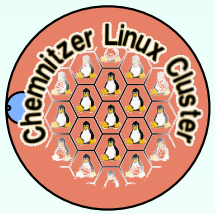
History of Peak Performance . . .

#CPUs :	32
MC-32:	160 Mflops
GC/PP:	10 Gflops
CLiC:	422 Gflops
CHiC:	8 Tflops

History of Peak Performance . . .

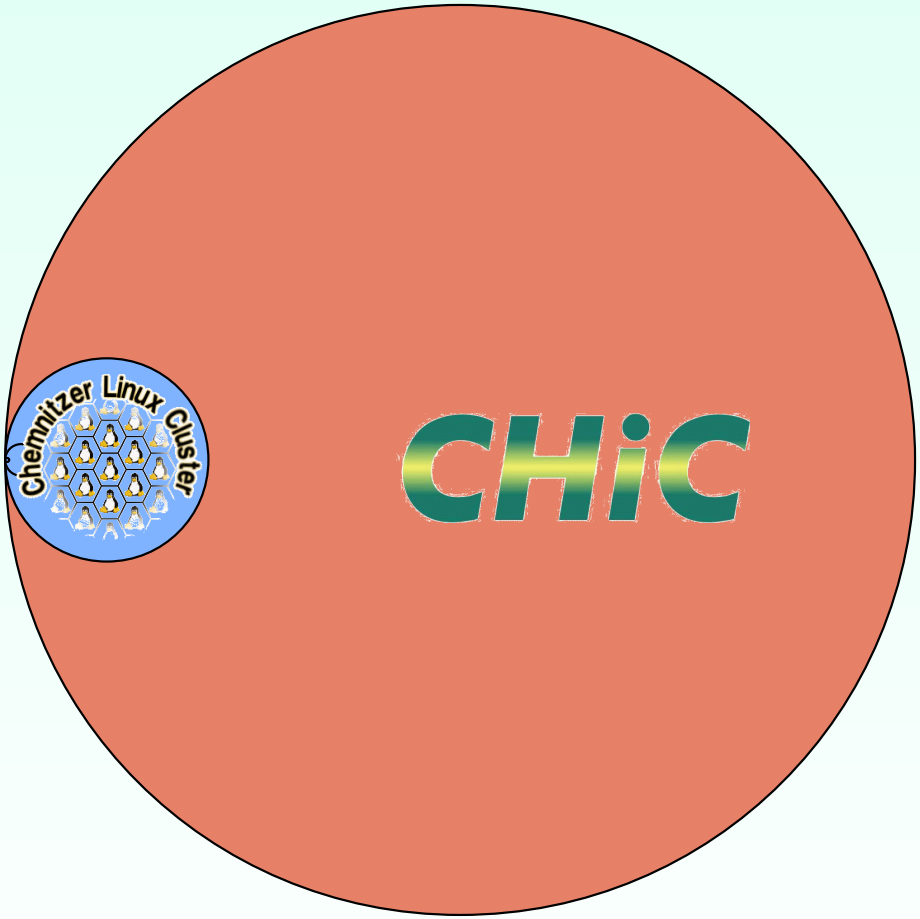
#CPUs :	128
MC-32:	160 Mflops
GC/PP:	10 Gflops
CLiC:	422 Gflops
CHiC:	8 Tflops

History of Peak Performance ...



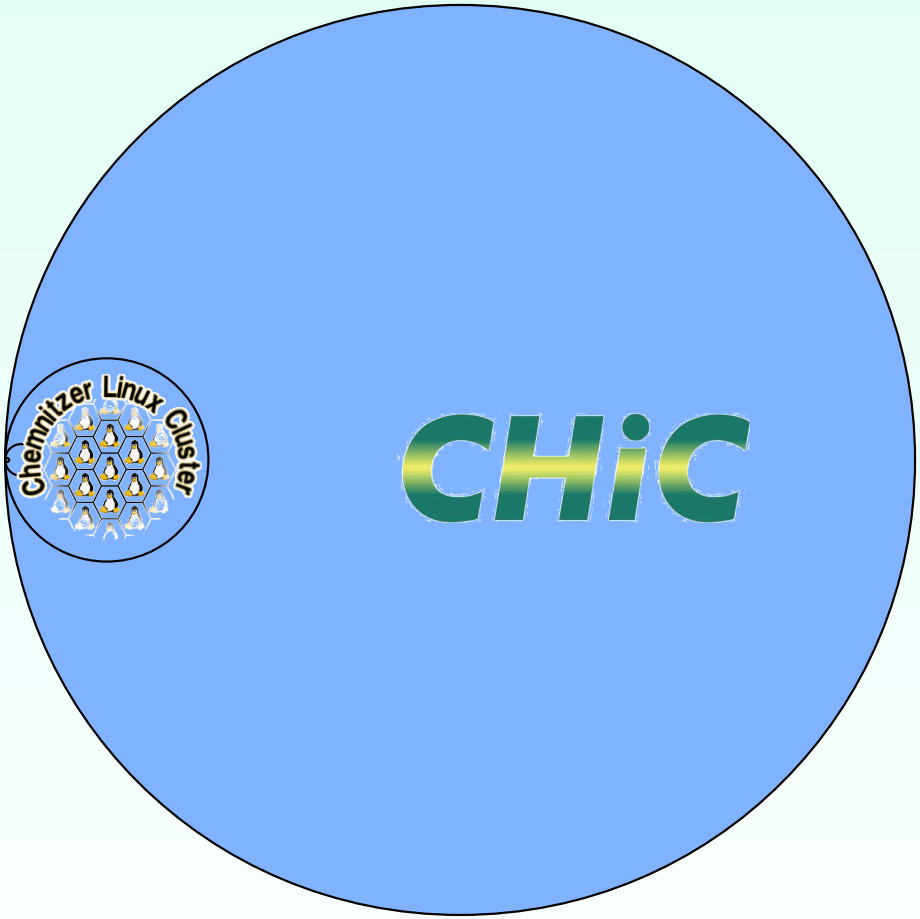
#CPUs :	528
MC-32:	160 Mflops
GC/PP:	10 Gflops
CLiC:	422 Gflops
CHiC:	8 Tflops

History of Peak Performance . . .



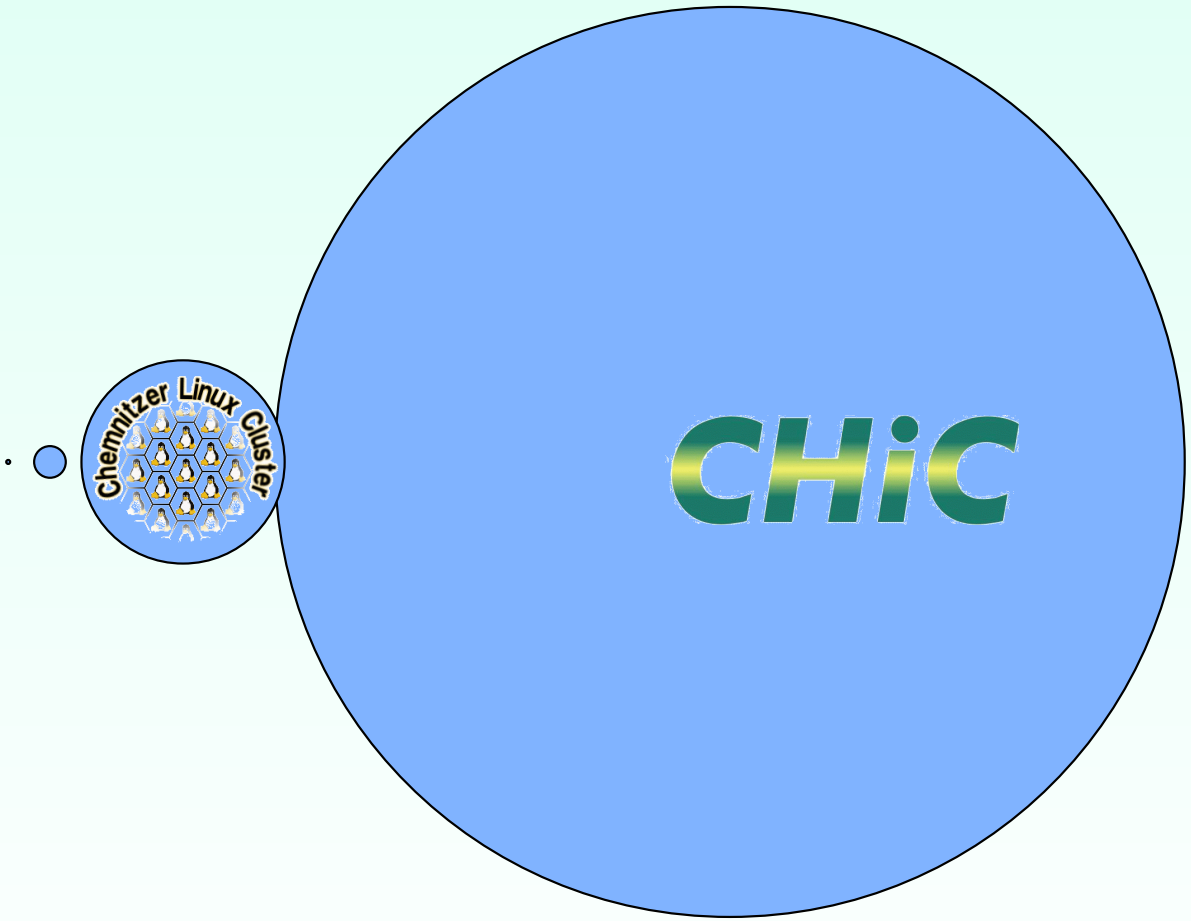
#CPUs :	535×4
MC-32:	160 Mflops
GC/PP:	10 Gflops
CLiC:	422 Gflops
CHiC:	8 Tflops

History of Peak Performance . . .

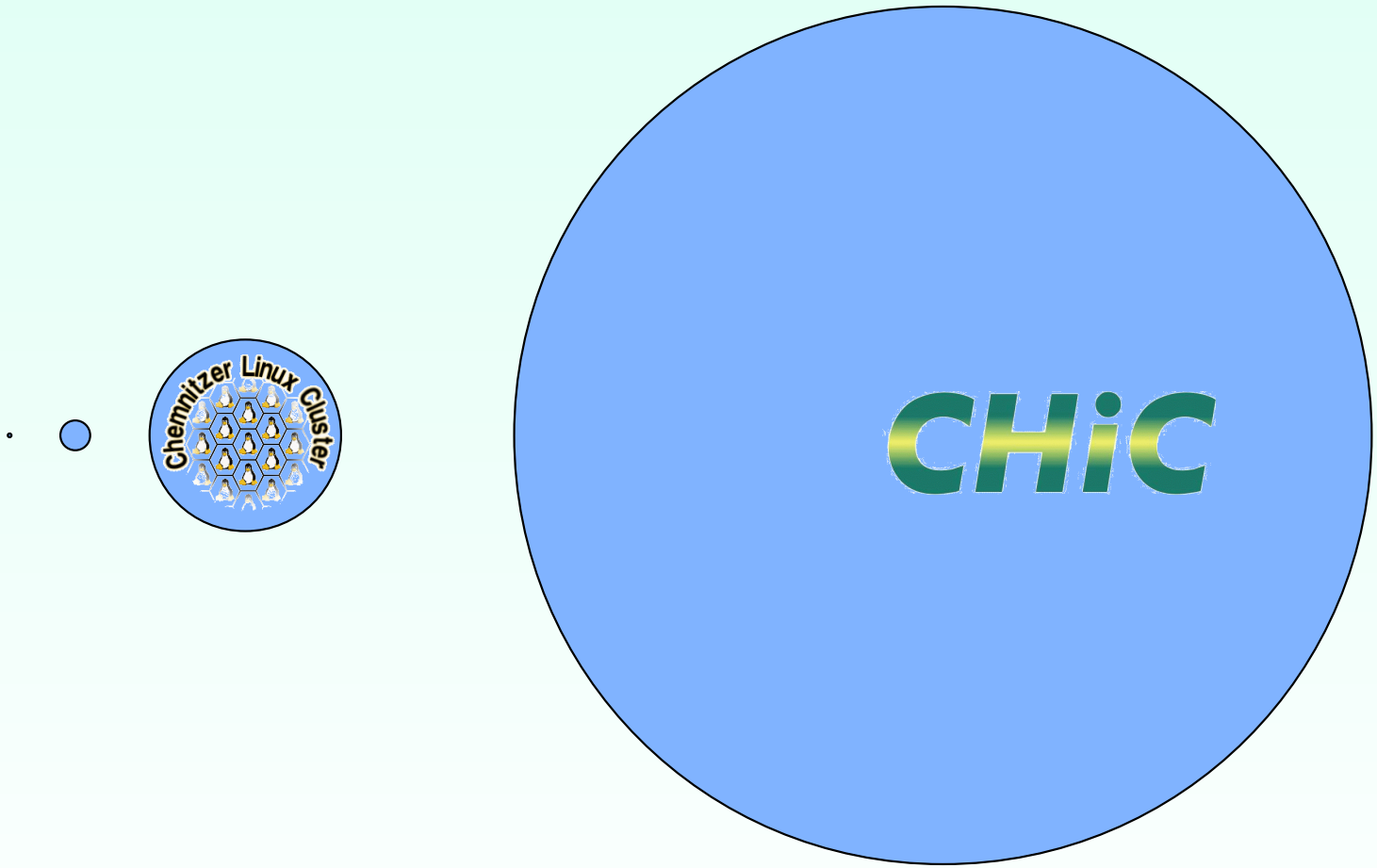


#CPUs :	535×4
MC-32:	160 Mflops
GC/PP:	10 Gflops
CLiC:	422 Gflops
CHiC:	8 Tflops

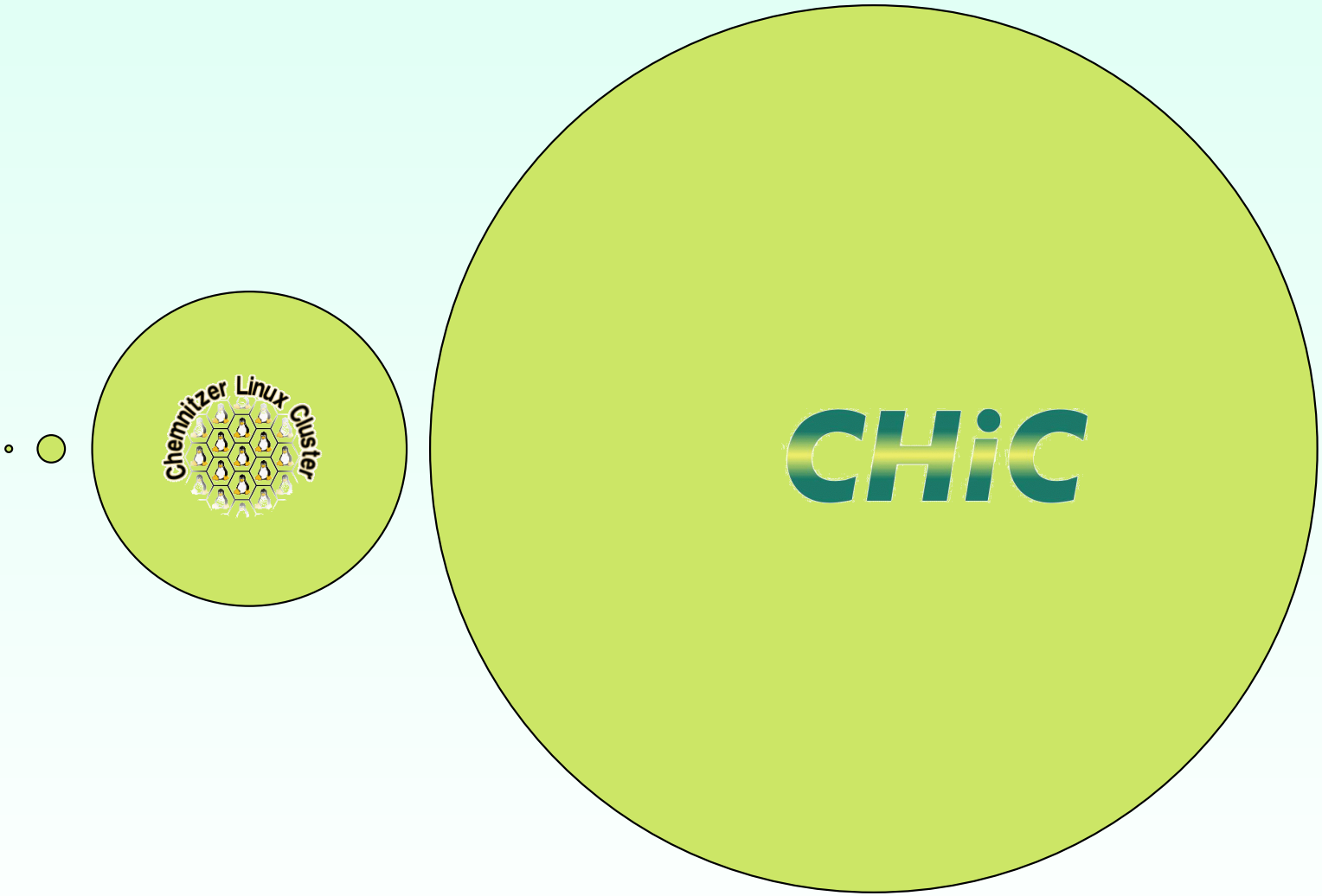
History of Peak Performance ...



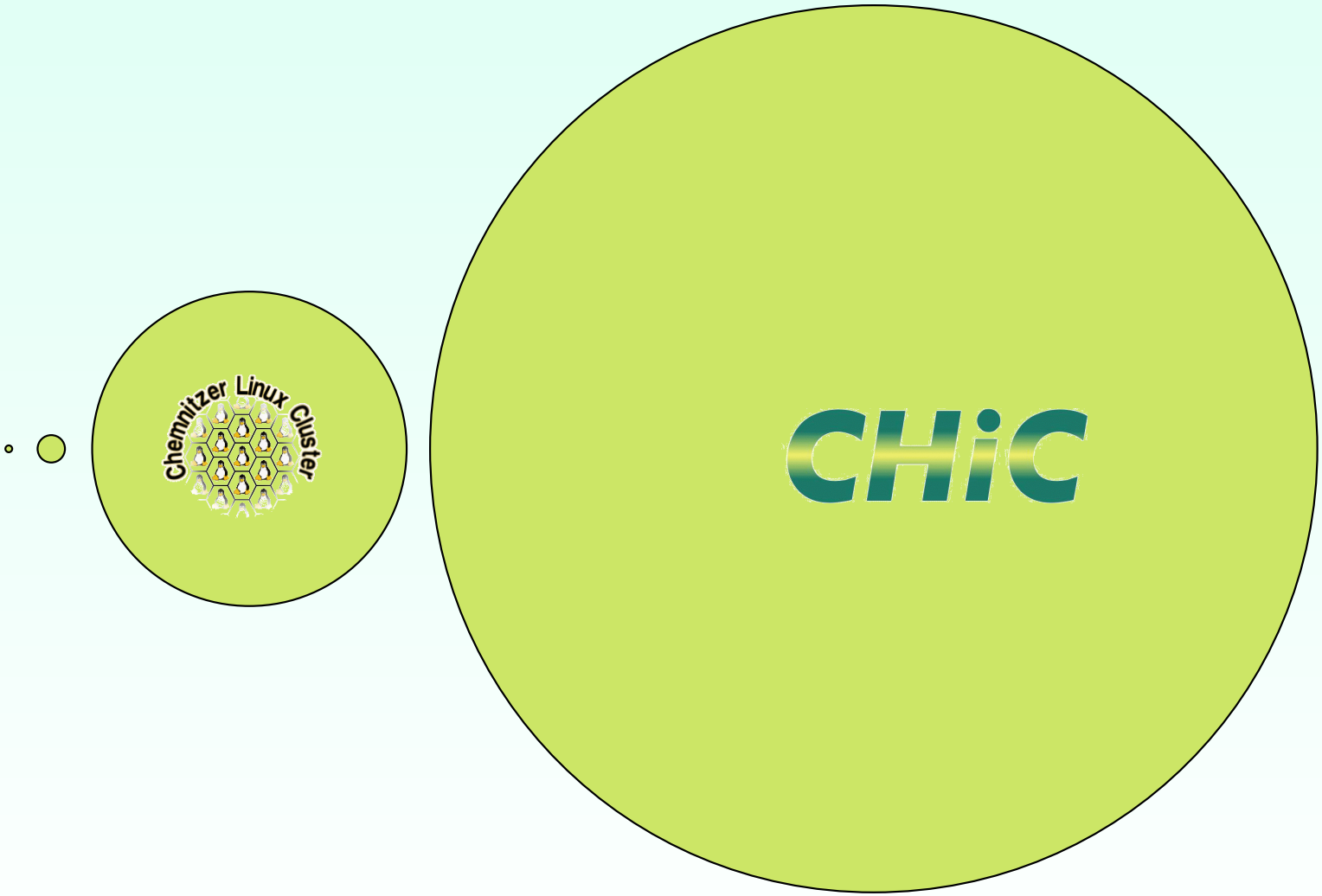
History of Peak Performance . . .



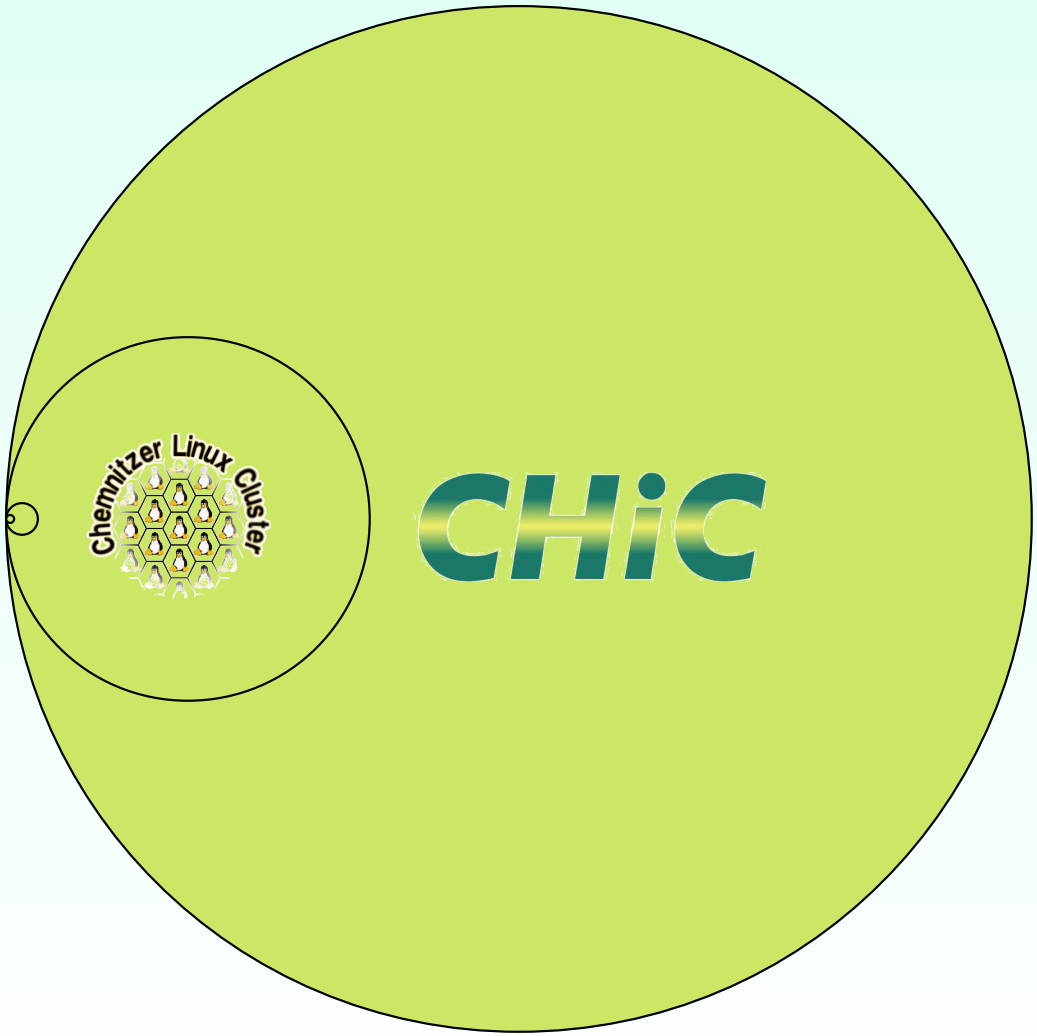
. . . and Working Memory



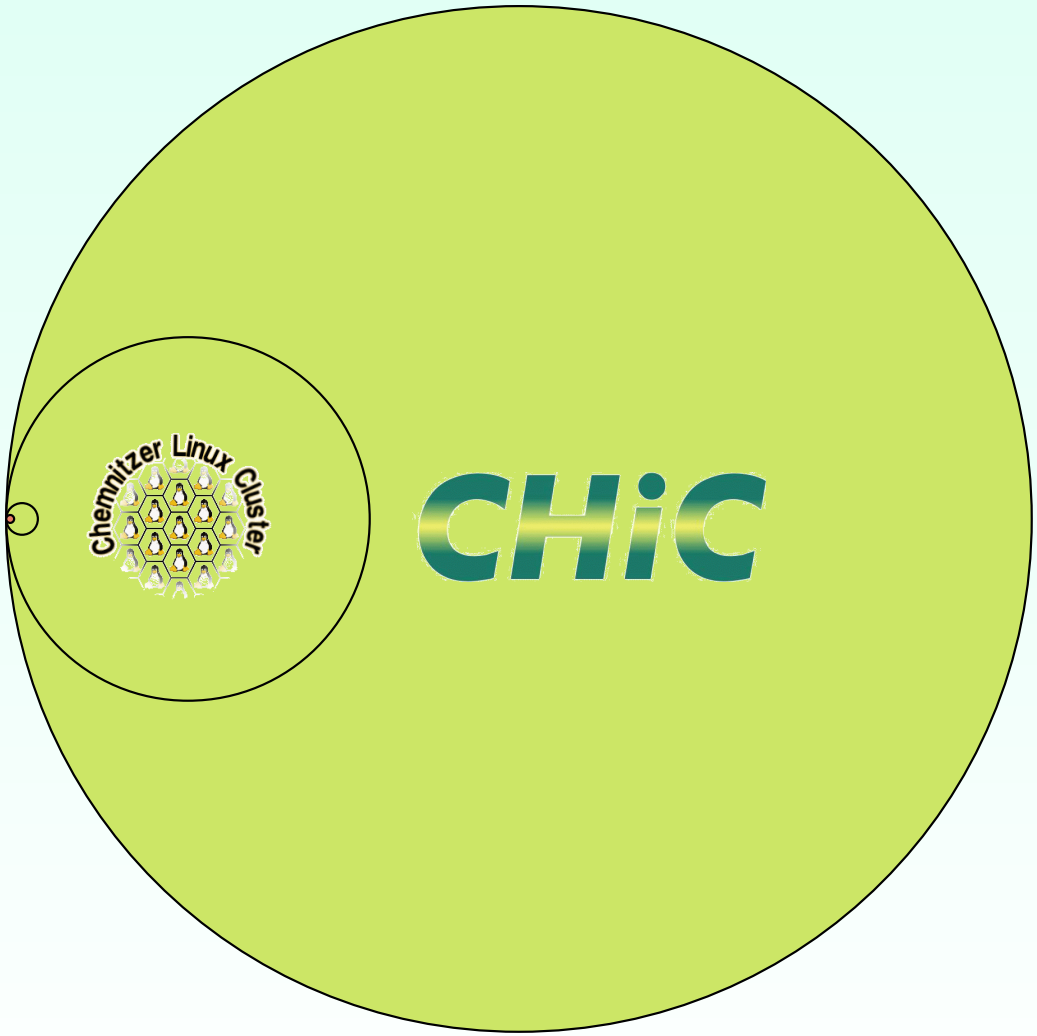
. . . and Working Memory



. . . and Working Memory

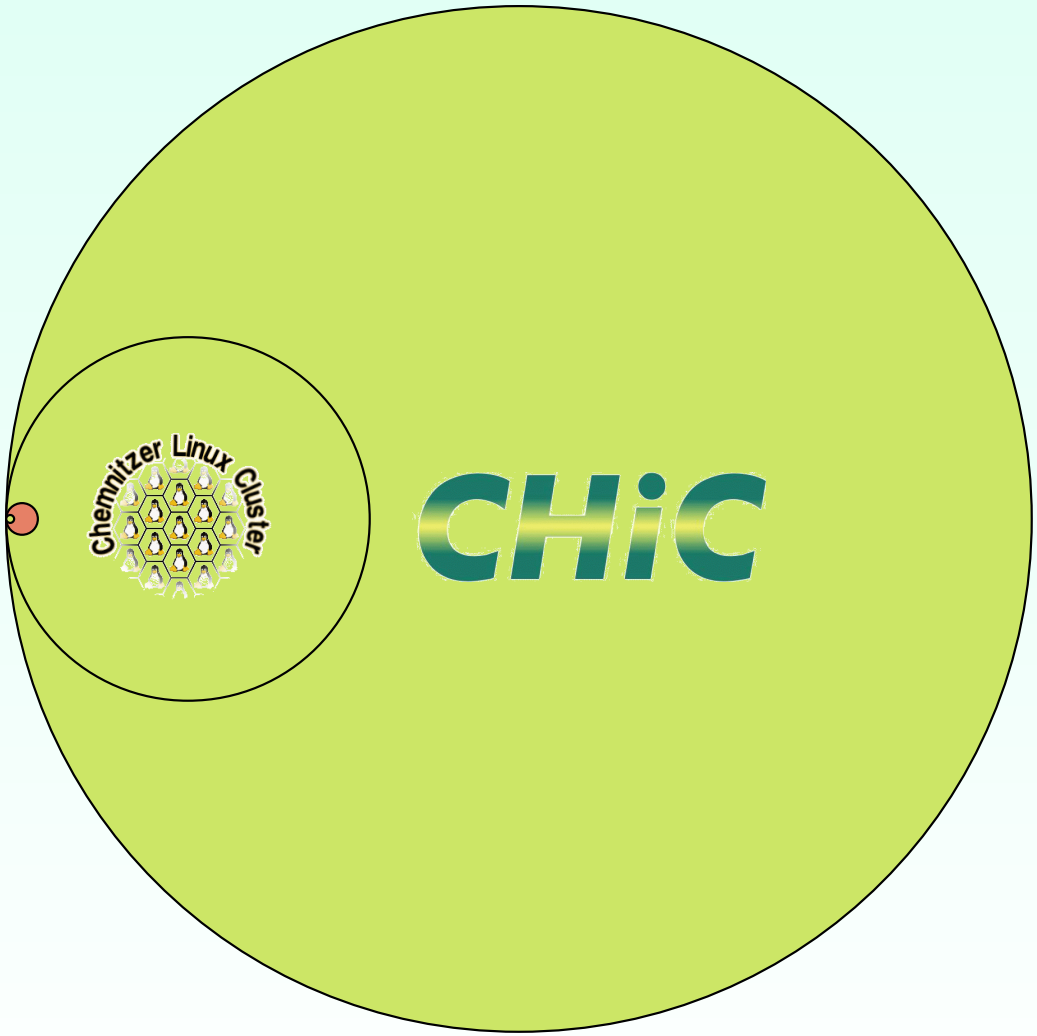


. . . and Working Memory



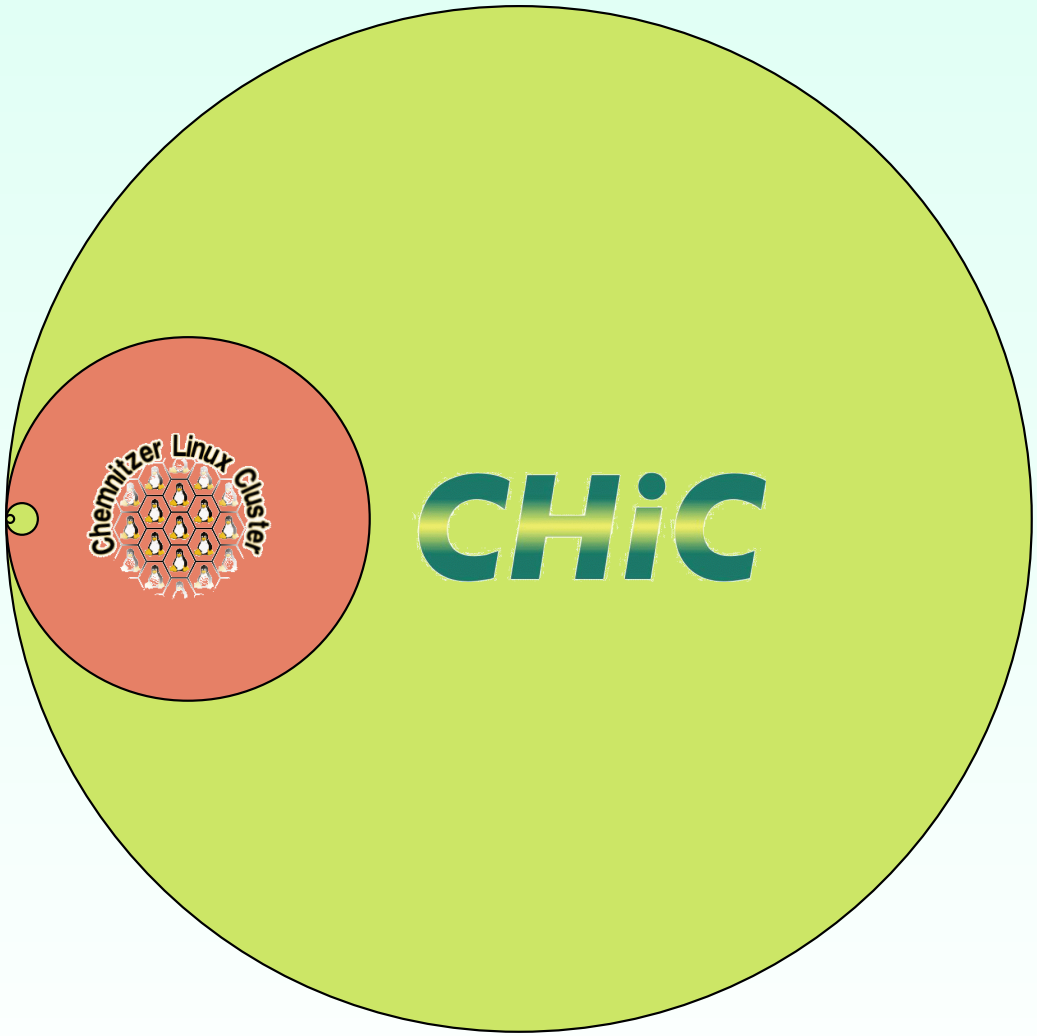
RAM	local:	4 MB
MC-32:		128 MB
GC/PP:		2 GB
CLiC:		270 GB
CHiC:		2 TB

. . . and Working Memory



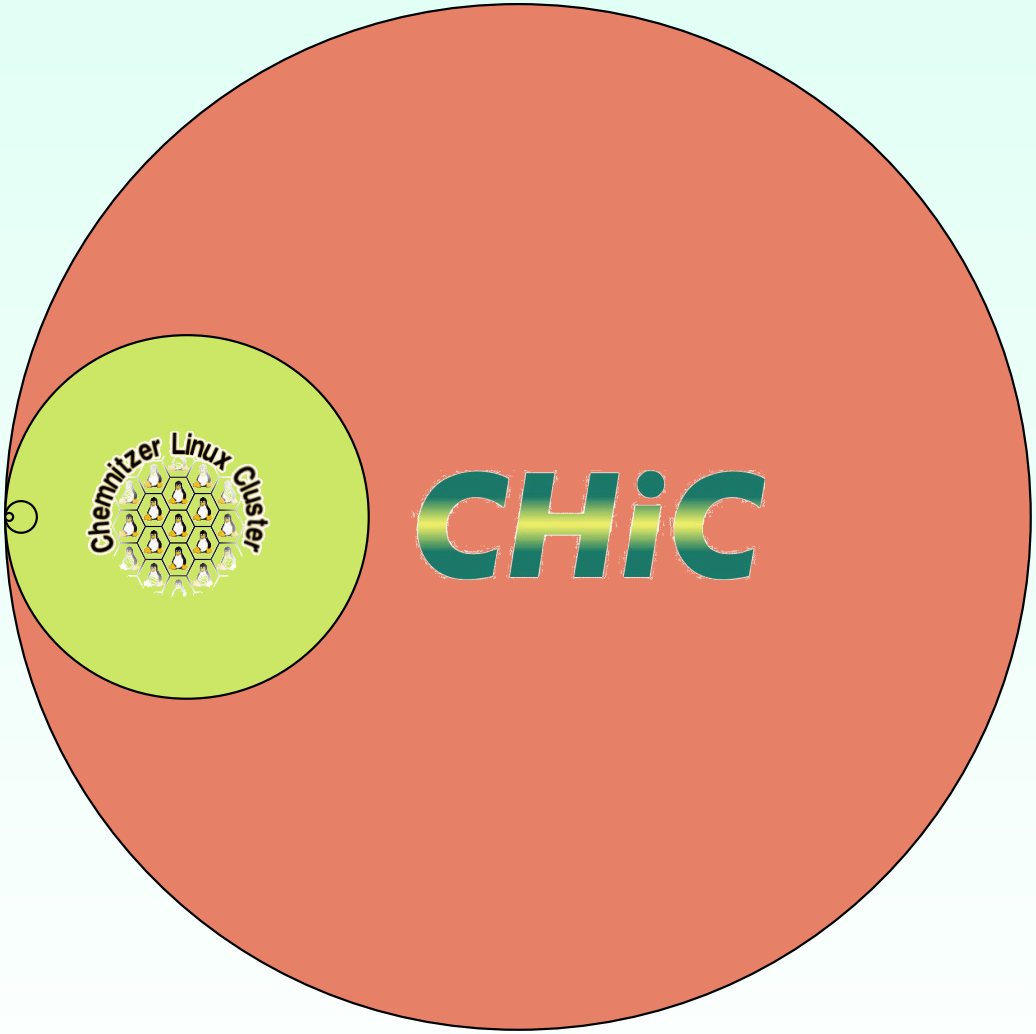
RAM	local:	16 MB
MC-32:		128 MB
GC/PP:		2 GB
CLiC:		270 GB
CHiC:		2 TB

. . . and Working Memory



RAM	local:	512 MB
MC-32:		128 MB
GC/PP:		2 GB
CLiC:		270 GB
CHiC:		2 TB

. . . and Working Memory



RAM	local:	4 GB
MC-32:		128 MB
GC/PP:		2 GB
CLiC:		270 GB
CHiC:		2 TB

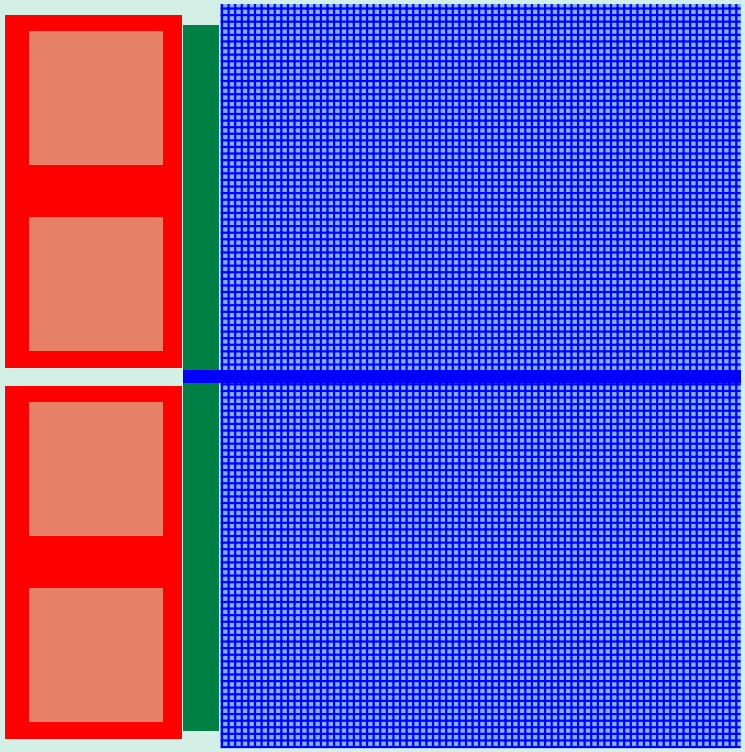
1. Chemnitz Super Computers
   - Hardware History
   - The Growth of Computing Power . . .
   - The Growth of Memory Capacity . . .

2. First Tests with Numerical Software
   - Hard- and Software Environment
   - Getting Access to the Cluster
   - Single Node Performance
   - Parallel Performance (8 Processors)
   - Global Communication

Test Environment

The Processor Nodes

2×AMD Opteron Dual Core



2 × 1 MB Cache 2 × 2 GB RAM

The Cluster

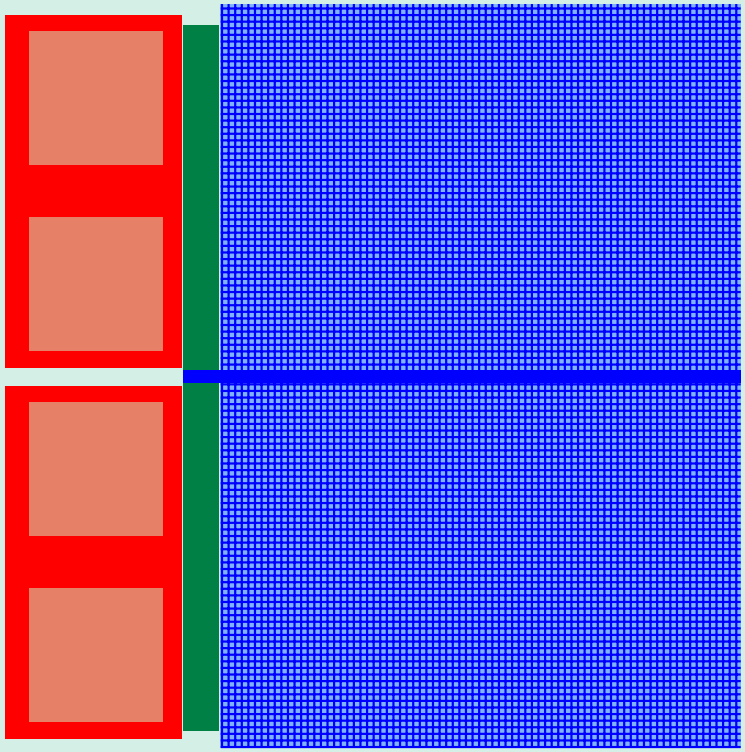
- 535 compute nodes (2.6 GHz),
  12 visualization nodes,
  8 I/O nodes,
  2 management and login nodes
- diskless, but high-performance parallel file access to a storage system ('lustre', 80 TB)
- highspeed interconnect technology InfiniBand
  (8...10 Gbit/s in Fortran)

The Problems ...

Test Environment

The Processor Nodes

2×AMD Opteron Dual Core



2 × 1 MB Cache 2 × 2 GB RAM

The Cluster

- 535 compute nodes (2.6 GHz),
  12 visualization nodes,
  8 I/O nodes,
  2 management and login nodes
- diskless, but high-performance parallel file access to a storage system ('lustre', 80 TB)
- highspeed interconnect technology InfiniBand
  (8...10 Gbit/s in Fortran)

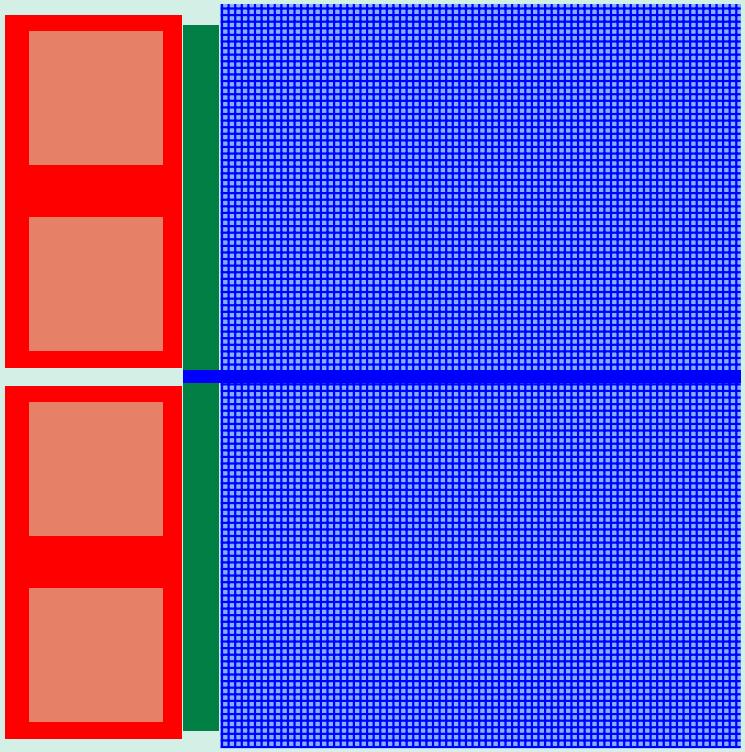
The Problems ...

Sensitivity of hardware and software

Test Environment

The Processor Nodes

2×AMD Opteron Dual Core



2 × 1 MB Cache 2 × 2 GB RAM

The Cluster

- 535 compute nodes (2.6 GHz),
  12 visualization nodes,
  8 I/O nodes,
  2 management and login nodes
- diskless, but high-performance parallel file access to a storage system ('lustre', 80 TB)
- highspeed interconnect technology InfiniBand
  (8...10 Gbit/s in Fortran)

The Problems ...

Sensitivity of hardware and software

System Software on CHiC

Multiple choice from different software packages ('modules'):

Compiling – 'comp/...'

- comp/gcc/346: g77, gcc, g++
- comp/gcc/422: gfortran, gcc, g++
- comp/path/31: pathf90, pathf95, pathcc, pathCC, pathdb
  EKOPath Compiler Suite with OpenMP support

Different MPI Implementations – 'mpi/...'

- mpi/openmpi/***
- mpi/mvapich2/***
- mpi/mpich2-tcp/***

... where '***' may be each of gcc346, gcc422 or path31

For compiling use always: mpicc / mpif77

System Software on CHiC

Multiple choice from different software packages ('modules'):

Compiling – 'comp/...'

- comp/gcc/346: g77, gcc, g++
- comp/gcc/422: gfortran, gcc, g++
- comp/path/31: pathf90, pathf95, pathcc, pathCC, pathdb
  EKOPath Compiler Suite with OpenMP support

Different MPI Implementations – 'mpi/...'

- mpi/openmpi/***
- mpi/mvapich2/***
- mpi/mpich2-tcp/***

... where '***' may be each of gcc346, gcc422 or path31

For compiling use always: mpicc / mpif77

System Software on CHiC

Multiple choice from different software packages ('modules'):

Compiling – 'comp/...'

- comp/gcc/346: g77, gcc, g++
- comp/gcc/422: gfortran, gcc, g++
- comp/path/31: pathf90, pathf95, pathcc, pathCC, pathdb
  EKOPath Compiler Suite with OpenMP support

Different MPI Implementations – 'mpi/...'

- mpi/openmpi/***
- mpi/mvapich2/***
- mpi/mpich2-tcp/***

... where '***' may be each of gcc346, gcc422 or path31

For compiling use always: mpicc / mpif77

System Software on CHiC

Multiple choice from different software packages ('modules'):

Mathematical Libraries – 'math/...'

- BLAS

  math/acml/gfortran64[_int64] (AMD Core Math Library)
  math/acml/pathscale64[_int64] [long integer versions]
  math/acml/3.6.0/gnu64 ...
  math/goto/gfortran-64[-int64] (Goto's Library)
  math/goto/g77-64[-int64]
  math/goto/pathscale-64[-int64] ...
- BLACS (math/blacs/***)
- LAPACK (math/lapack/***)
- SCALAPACK (math/scalapack/***)

  (each in multiple versions for comp and mpi)

System Software on CHiC

Multiple choice from different software packages ('modules'):

Mathematical Libraries – 'math/...'

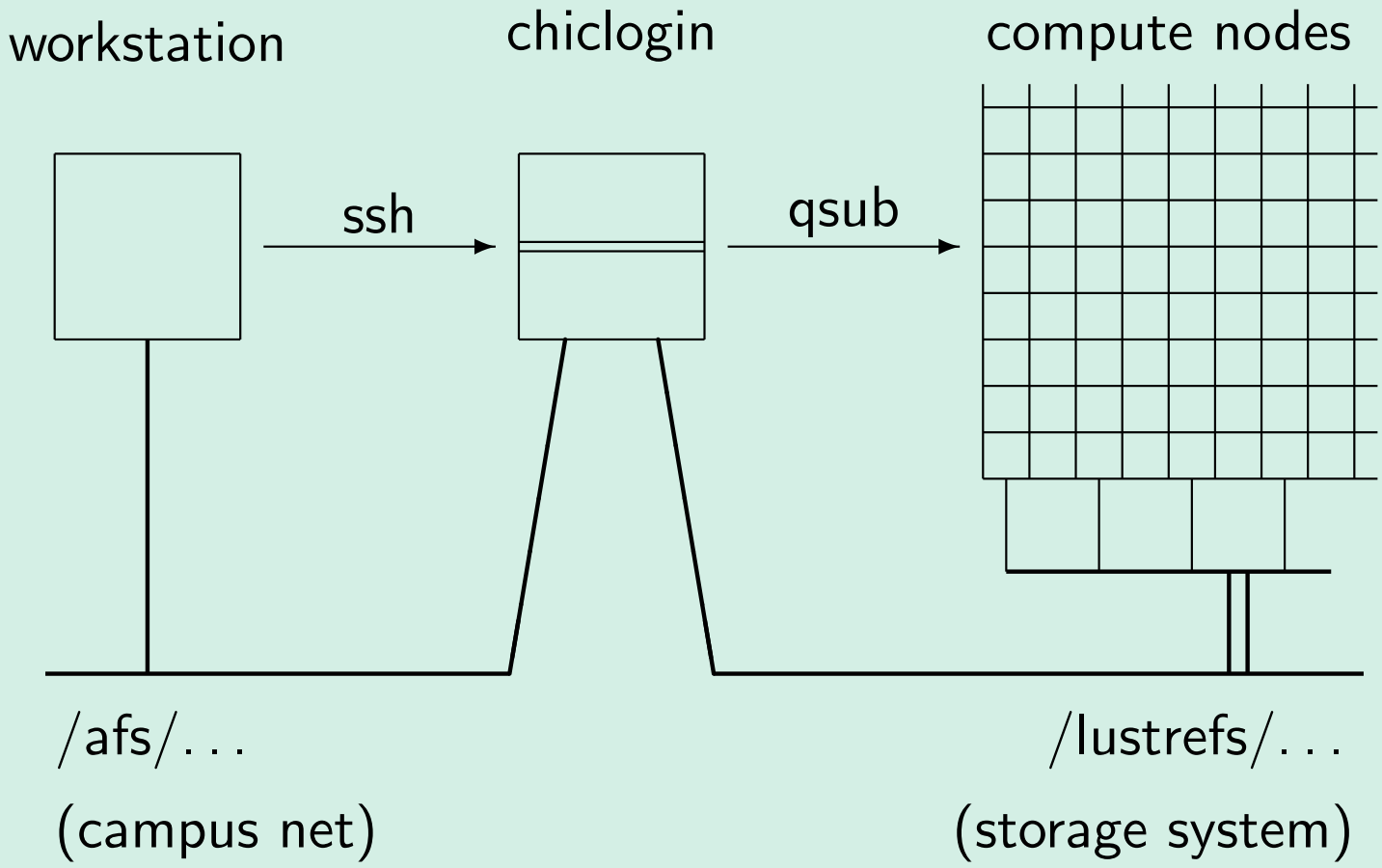
- BLAS

  `math/acml/gfortran64[_int64]` (AMD Core Math Library)
  `math/acml/pathscale64[_int64]` [long integer versions]
  `math/acml/3.6.0/gnu64` ...
  `math/goto/gfortran-64[-int64]` (Goto's Library)
  `math/goto/g77-64[-int64]`
  `math/goto/pathscale-64[-int64]` ...
- BLACS (`math/blacs/***`)
- LAPACK (`math/lapack/***`)
- SCALAPACK (`math/scalapack/***`)

  (each in multiple versions for comp and mpi)

Getting Started

Access to the Cluster



Getting Started

Job Queues (TORQUE/Maui ← OpenPBS)

- Interactive jobs (usually with a small number of nodes), e.g.

  ```
  qsub -I -l nodes=8:compute:ppn=2,walltime=00:30:00
  ```

  means: get 8 compute nodes, intended to run 2 processes per node for not more than 30 minutes in interactive mode.

- Batch jobs (for expensive, lengthy or unamusing computations), Command line arguments of qsub may be part of the script to be submitted (as special comments), e.g.

  ```
  #!/bin/sh
  #PBS -l nodes=64:compute:ppn=1,walltime=4:00:00,mem=2gb
  #PBS -A <project_account>
  #PBS -W x=NACCESSPOLICY:SINGLEJOB
  ```

  Submit the batch job: `qsub <scriptfile>`

Getting Started

Job Queues (TORQUE/Maui ← OpenPBS)

- Interactive jobs (usually with a small number of nodes), e.g.

  ```
  qsub -I -l nodes=8:compute:ppn=2,walltime=00:30:00
  ```

  means: get 8 compute nodes, intended to run 2 processes per node for not more than 30 minutes in interactive mode.

- Batch jobs (for expensive, lengthy or unamusing computations), Command line arguments of qsub may be part of the script to be submitted (as special comments), e.g.

  ```
  #!/bin/sh
  #PBS -l nodes=64:compute:ppn=1,walltime=4:00:00,mem=2gb
  #PBS -A <project_account>
  #PBS -W x=NACCESSPOLICY:SINGLEJOB
  ```

  Submit the batch job: `qsub <scriptfile>`

Getting Started

Job Queues (TORQUE/Maui ← OpenPBS)

- Current configuration of job queues:

short	≤30 min	≤512 nodes
medium	≤ 4 h	≤256 nodes
long	≤ 48 h	≤128 nodes
verylong	≤ 720 h	≤ 64 nodes

- Special options

  `#PBS -W x=NACCESSPOLICY:SINGLEJOB`
  necessary for exclusive node access, otherwise the nodes may be shared with other users.

  `#PBS -l nodes=1:bigmem:ppn=1+15:compute:ppn=1,...`
  for interactive jobs with graphical output from node 0
  (‘`bigmem`’ implies that X11 is available on the node)

Getting Started

Job Queues (TORQUE/Maui ← OpenPBS)

- Current configuration of job queues:

short	≤30 min	≤512 nodes
medium	≤ 4 h	≤256 nodes
long	≤ 48 h	≤128 nodes
verylong	≤ 720 h	≤ 64 nodes

- Special options

  `#PBS -W x=NACCESSPOLICY:SINGLEJOB`

  necessary for exclusive node access, otherwise the nodes may be shared with other users.

  `#PBS -l nodes=1:bigmem:ppn=1+15:compute:ppn=1,...`

  for interactive jobs with graphical output from node 0 (‘`bigmem`’ implies that X11 is available on the node)

My Ordinary Cluster Tests

Test Situations

- Single processor = one node, only one CPU (of '4')
- `mpirun -np 64 ...`
  - 64 nodes, (ppn=1, upto 2 GByte)
  - 32 nodes, (ppn=2, upto 1.5 GByte)
  - 16 nodes, (ppn=4, $<$ 1 GByte)
- 2 alternate MPI versions (MVAPICH, Open MPI)
- 2 alternate private communication libraries
  - `MPIcom` - global communication using `MPI_Allreduce` etc.
  - `MPIcubecom` - hypercube-mode with only `MPI_sendrecv`

- Time measurement could be complicated by running or hanging processes – own or others. Now this should be excluded by the batch system (but not really sure).

My Ordinary Cluster Tests

Test Situations

- Single processor = one node, only one CPU (of '4')
- `mpirun -np 64 ...`
  - 64 nodes, (ppn=1, upto 2 GByte)
  - 32 nodes, (ppn=2, upto 1.5 GByte)
  - 16 nodes, (ppn=4, $<$ 1 GByte)
- 2 alternate MPI versions (MVAPICH, Open MPI)
- 2 alternate private communication libraries
  - `MPIcom` - global communication using `MPI_Allreduce` etc.
  - `MPIcubecom` - hypercube-mode with only `MPI_sendrecv`

- Time measurement could be complicated by running or hanging processes – own or others. Now this should be excluded by the batch system (but not really sure).

Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const.$
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const$.
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

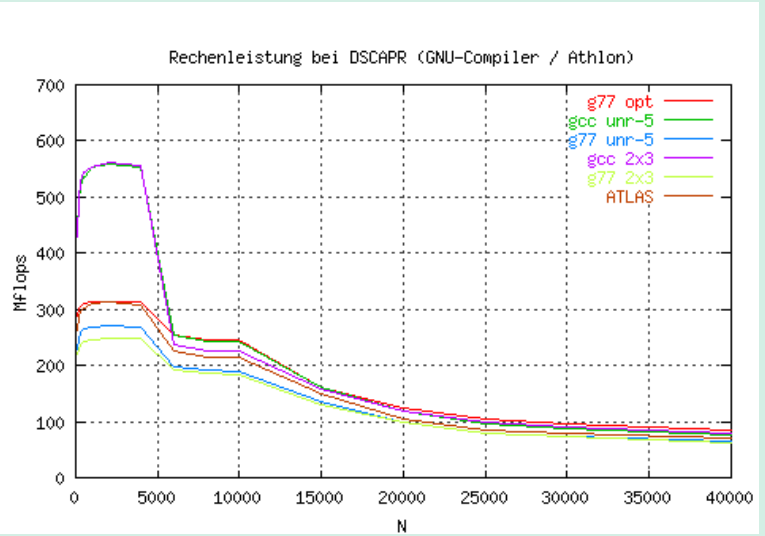
Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const.$
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

Athlon-500 (GNU-Compiler)



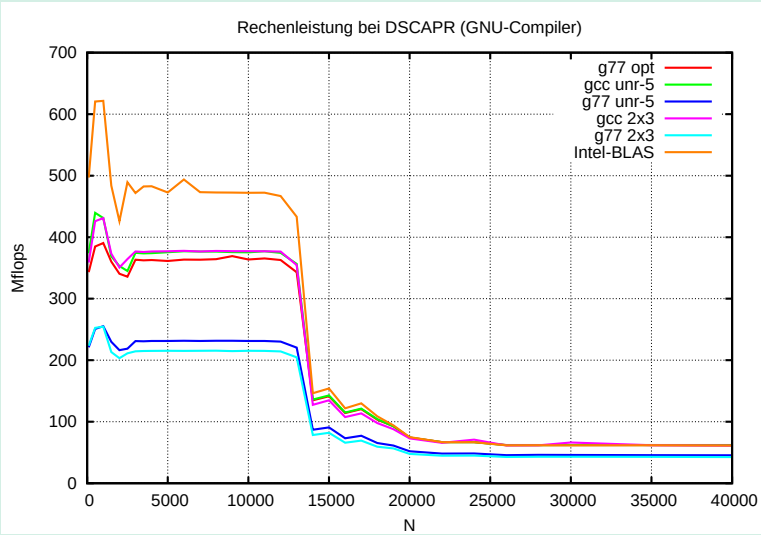
Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const$.
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

P III-800 (GNU-Compiler)



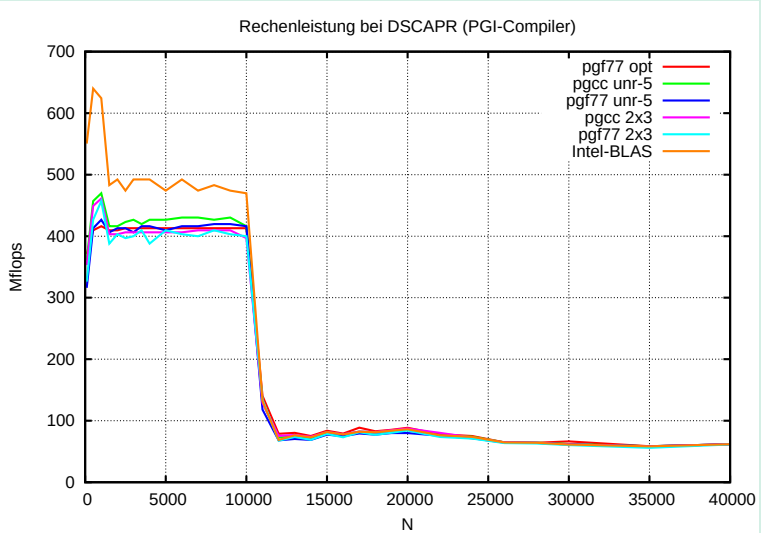
Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const$.
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

P III-800 (PGI-Compiler-Suite)



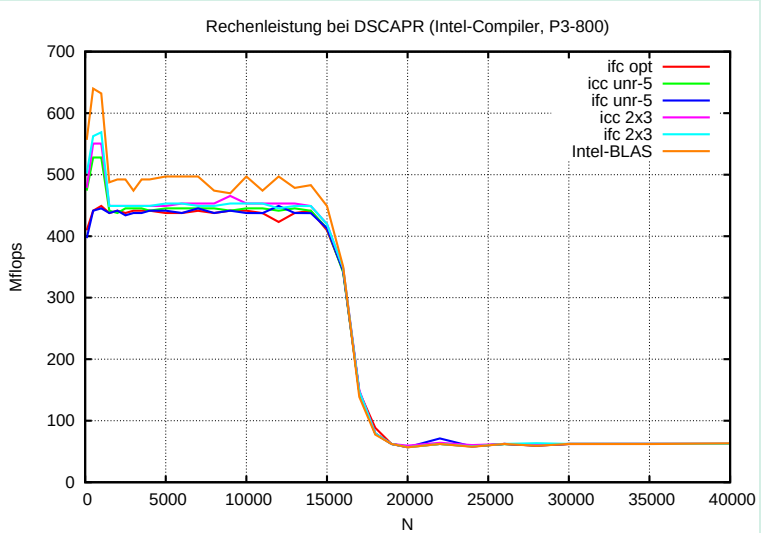
Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const.$
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

P III-800 (Intel-Compiler-Suite)



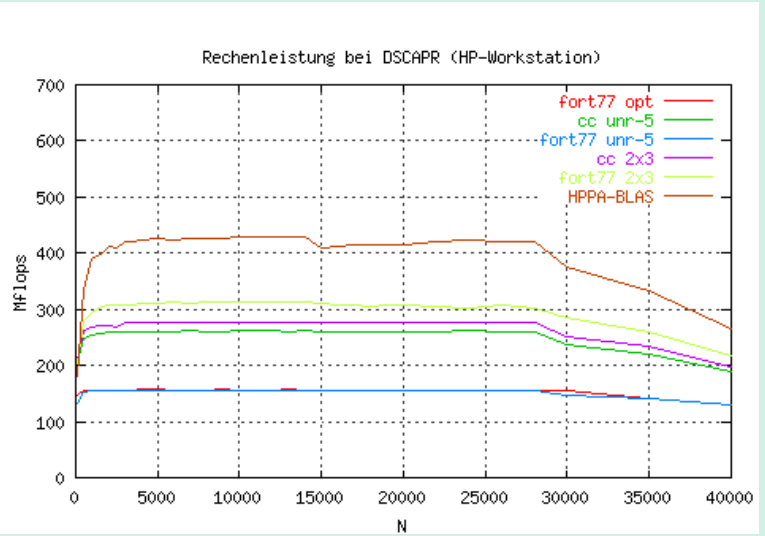
Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const.$
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

HP Workstation



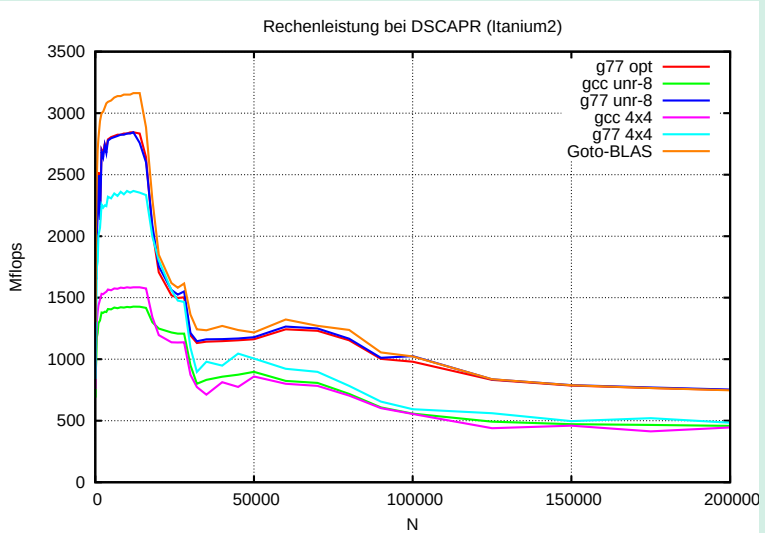
Single Node Performance

(1) Computing dot products

- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const$.
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

For comparison:

Itanium-2 (GNU und Goto-BLAS)



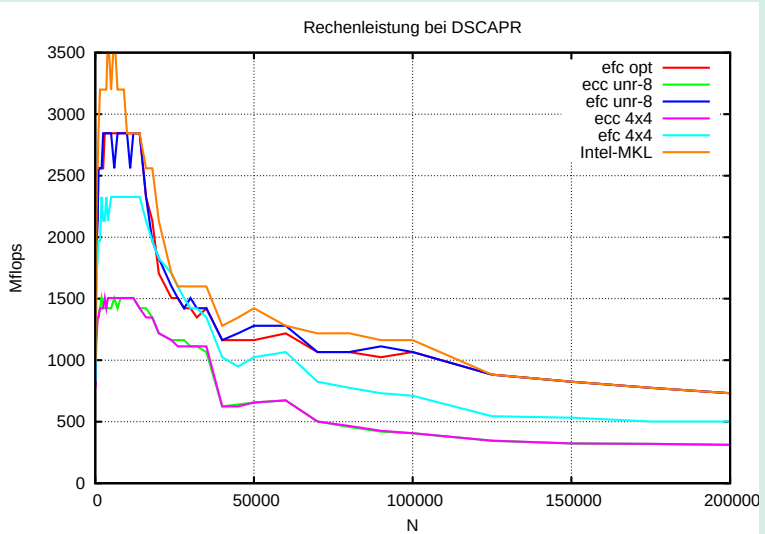
Single Node Performance

(1) Computing dot products

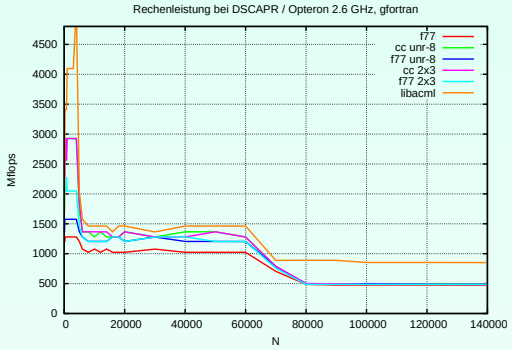
- Compute (k_N-times) $s = \sum_{i=1}^{N} x_i y_i$
- for varying vector length $N = 100, \ldots, 100\,000, \ldots,$ and $k_N \cdot N \approx const.$
- different program versions (simple, unrolled loops; C, Fortran)
- Mflops determined from computing time, showing
- dependency on memory access (for small N almost only cache)

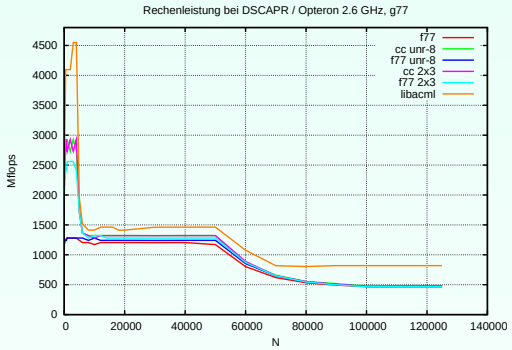
For comparison:

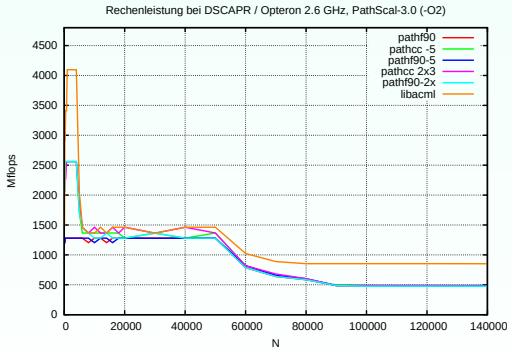
Itanium-2 (Intel-Comp. und MKL)



Single Node Performance on CHiC

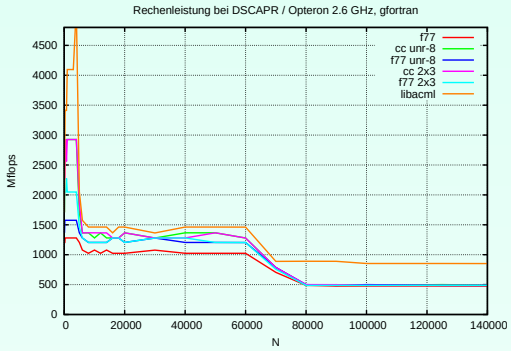


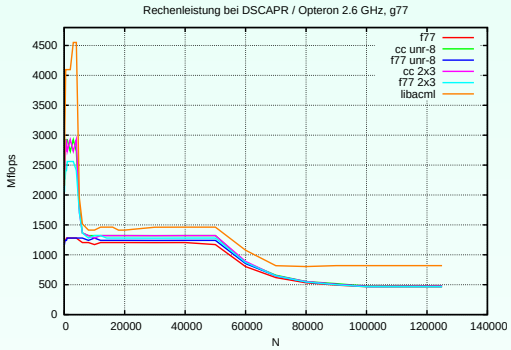


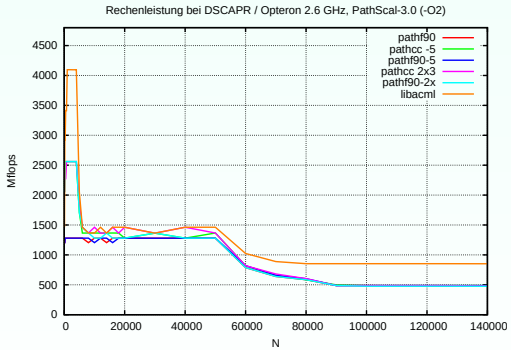


CHiC –

Single Node Performance on CHiC

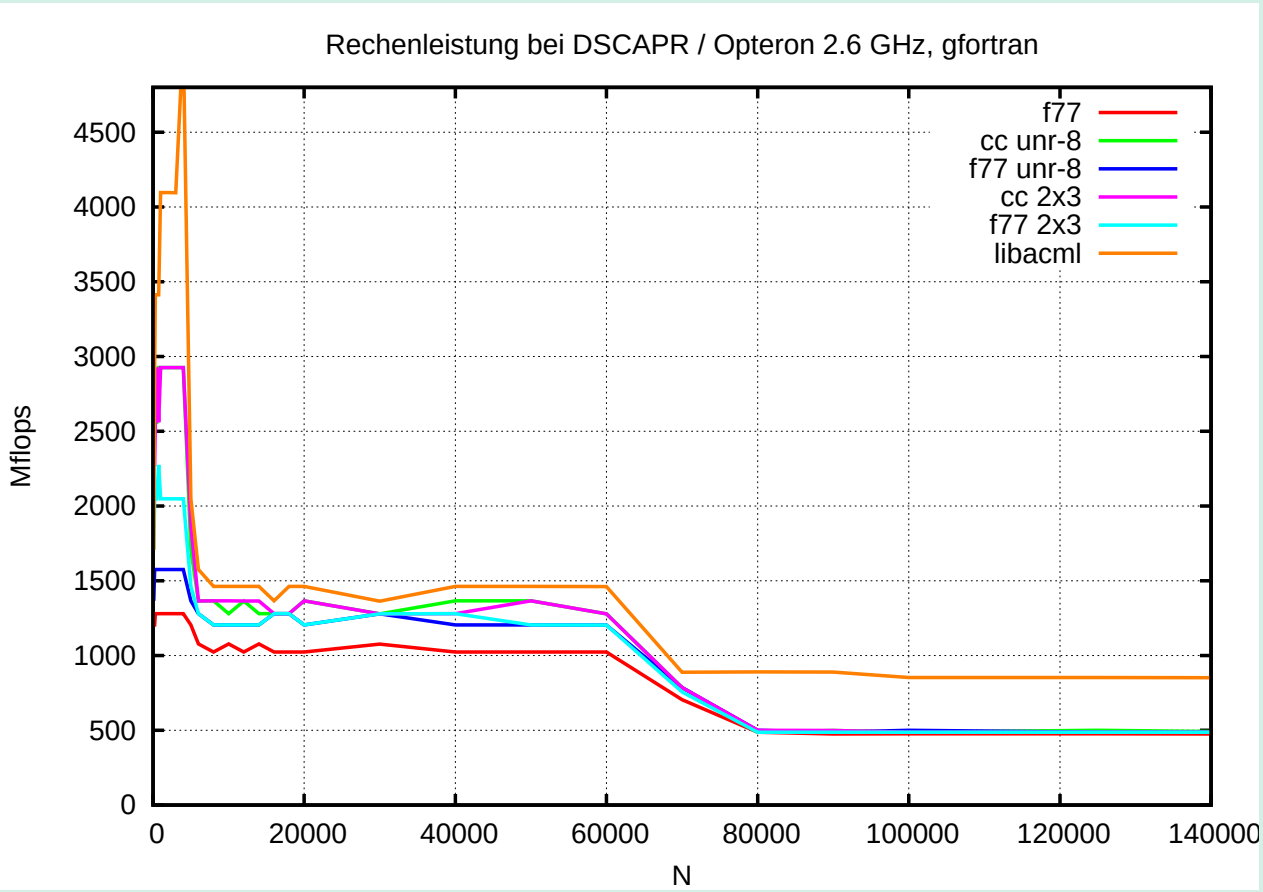




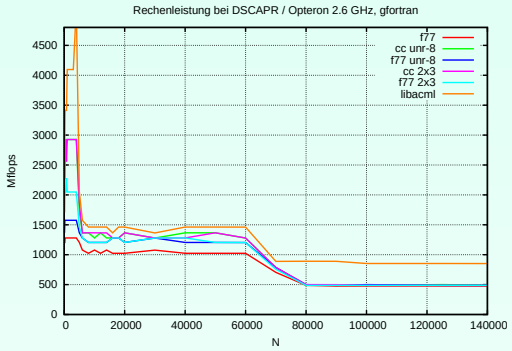


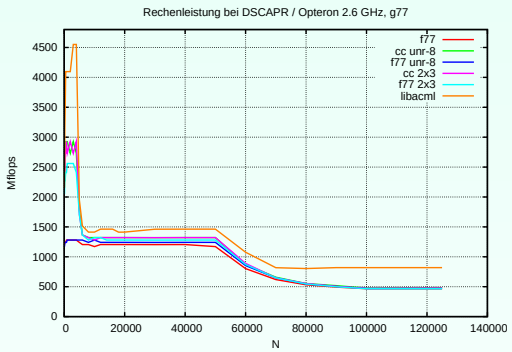
BLAS library: ACML = AMD Core Math Library

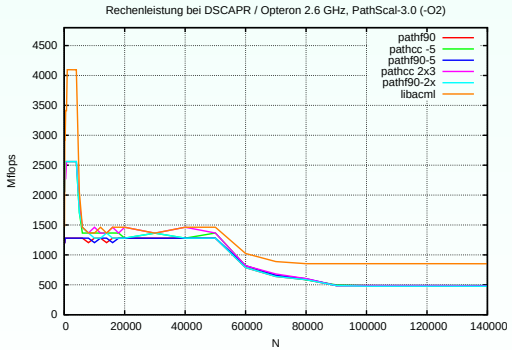
CHiC – gfortran/gcc



Single Node Performance on CHiC

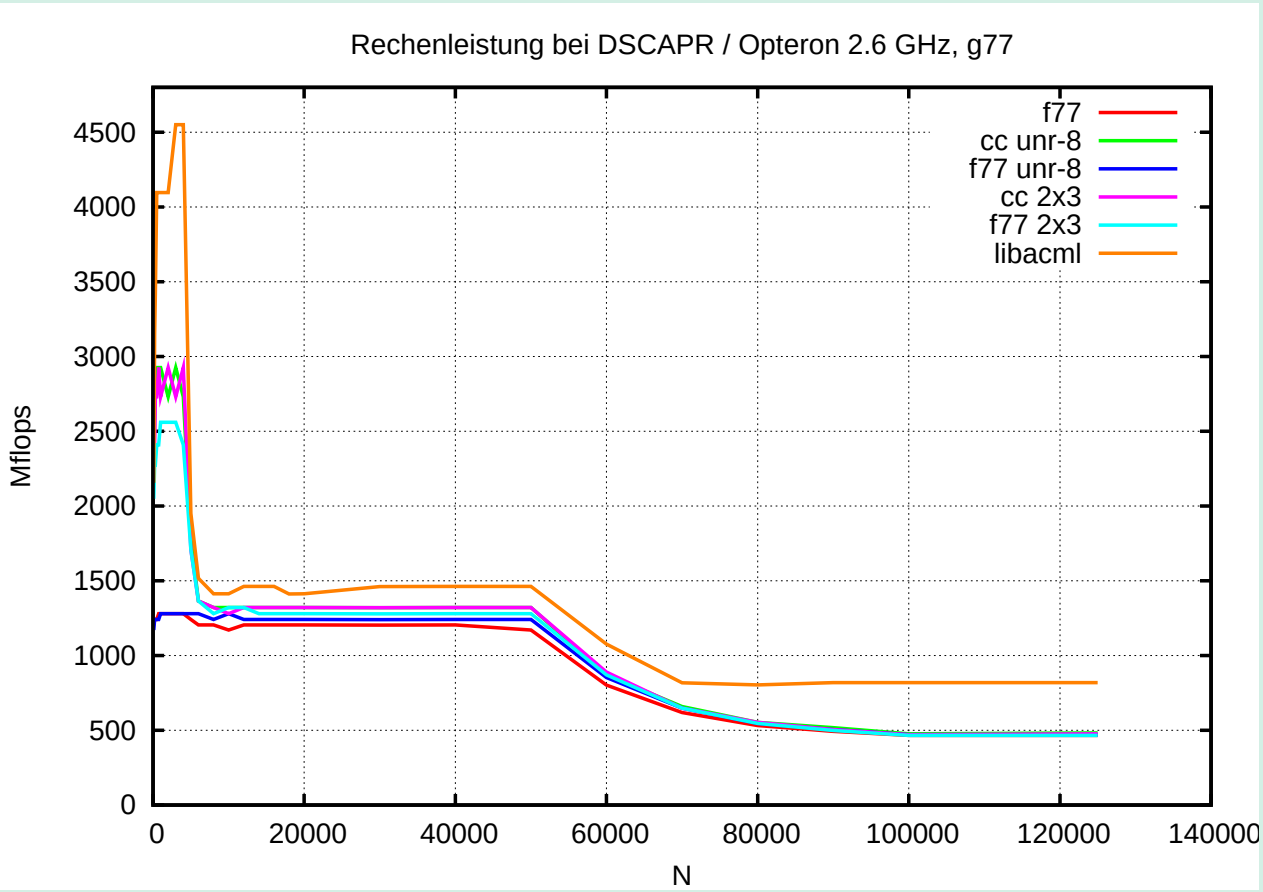




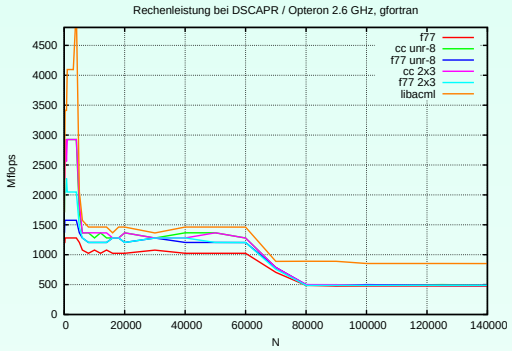


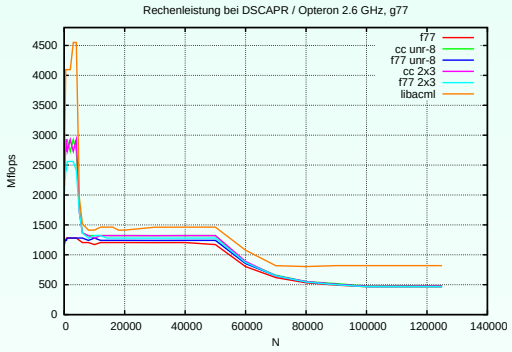
BLAS library: ACML = AMD Core Math Library

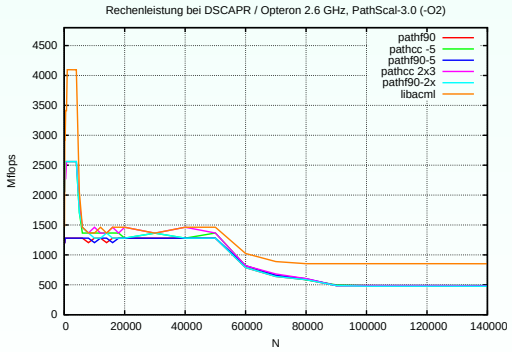
CHiC – g77/gcc



Single Node Performance on CHiC

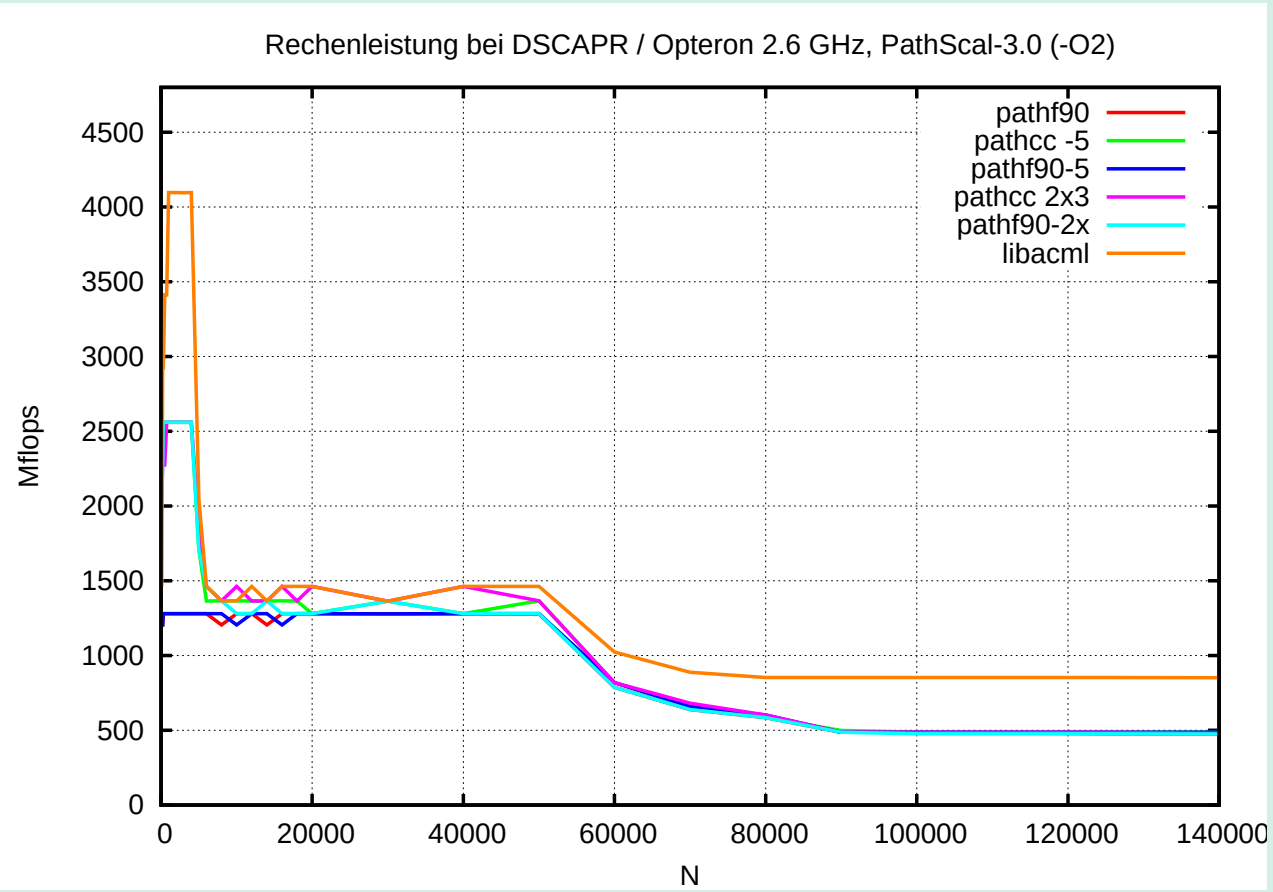




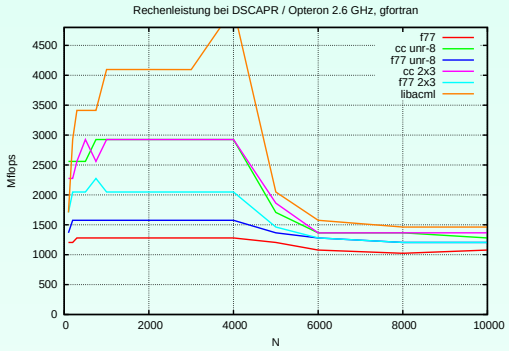


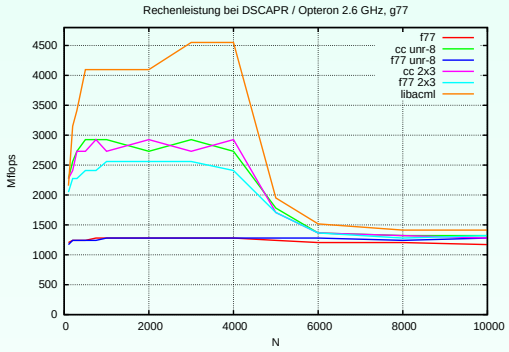
BLAS library: ACML = AMD Core Math Library

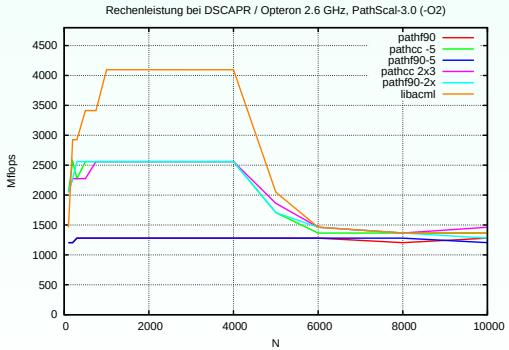
CHiC – PathScale-3.0 (-O2)



Single Node Performance on CHiC

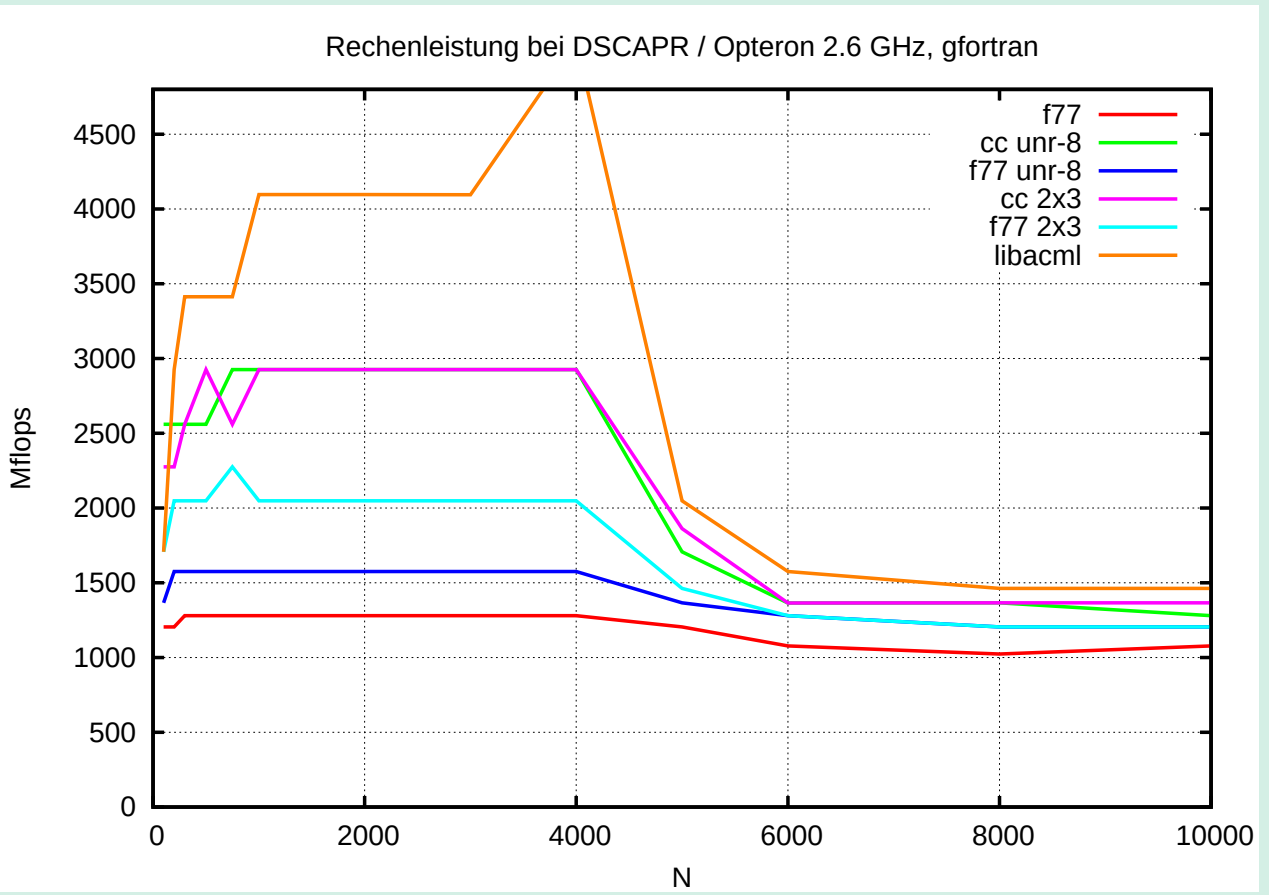




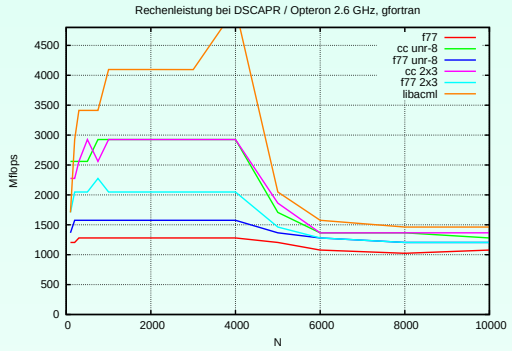


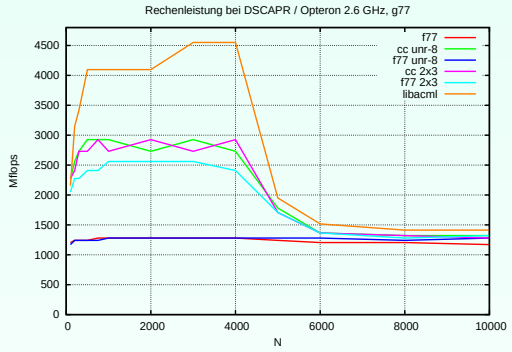
BLAS library: ACML = AMD Core Math Library

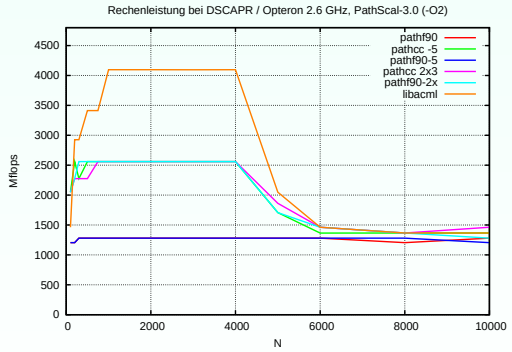
CHiC – gfortran/gcc



Single Node Performance on CHiC

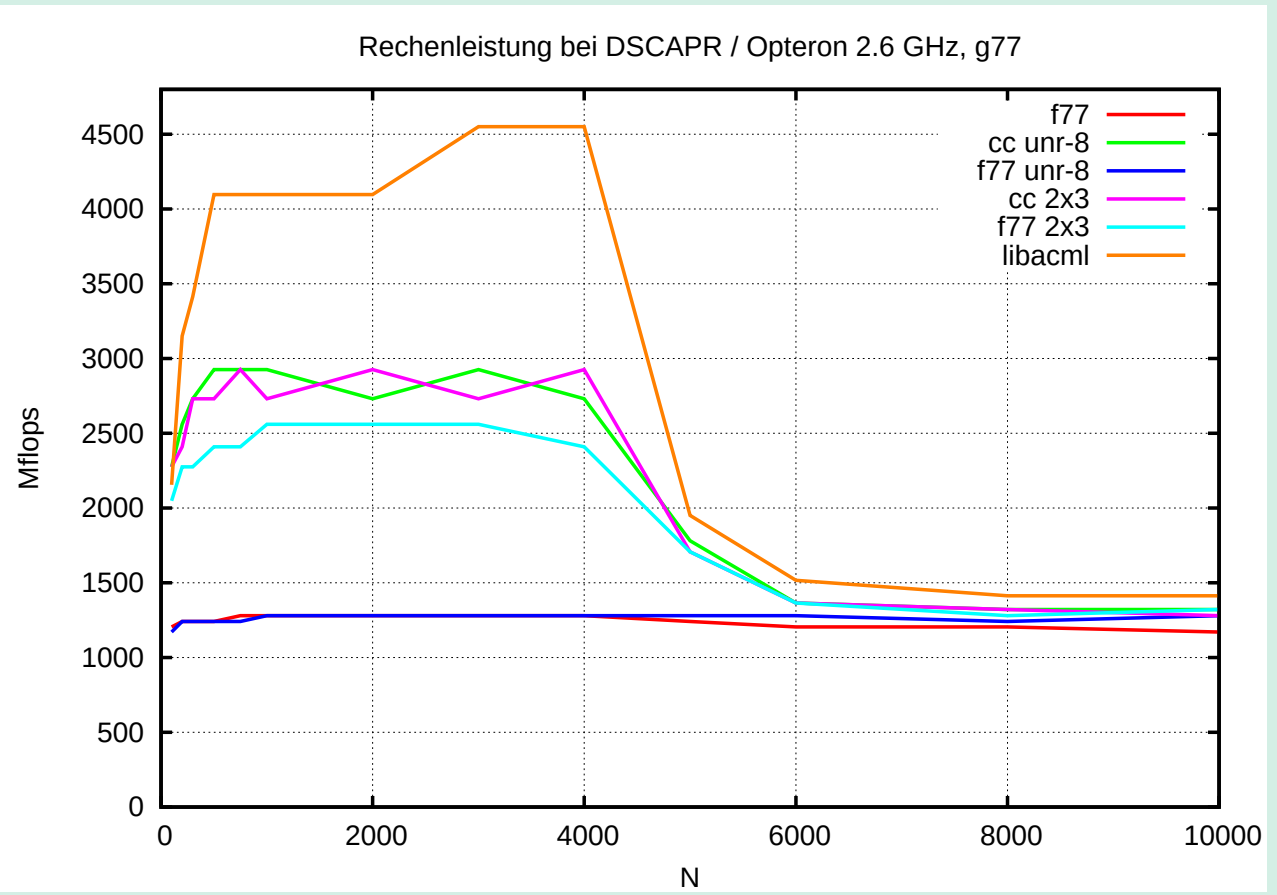




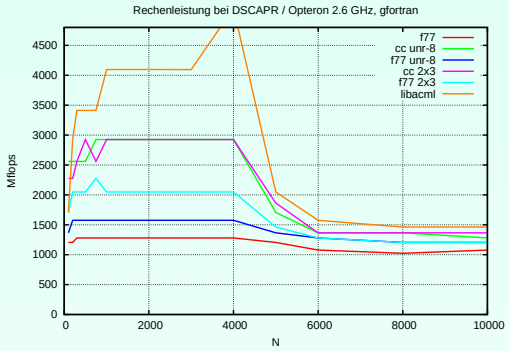


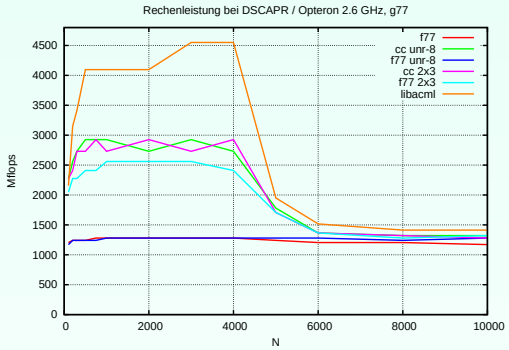
BLAS library: ACML = AMD Core Math Library

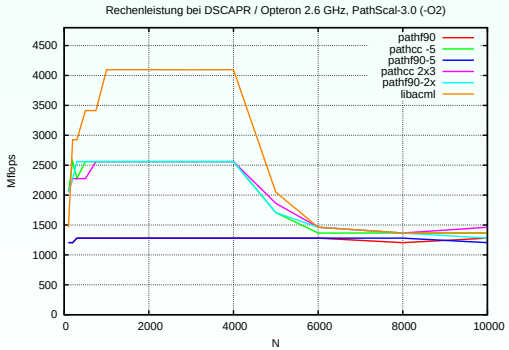
CHiC – g77/gcc



Single Node Performance on CHiC

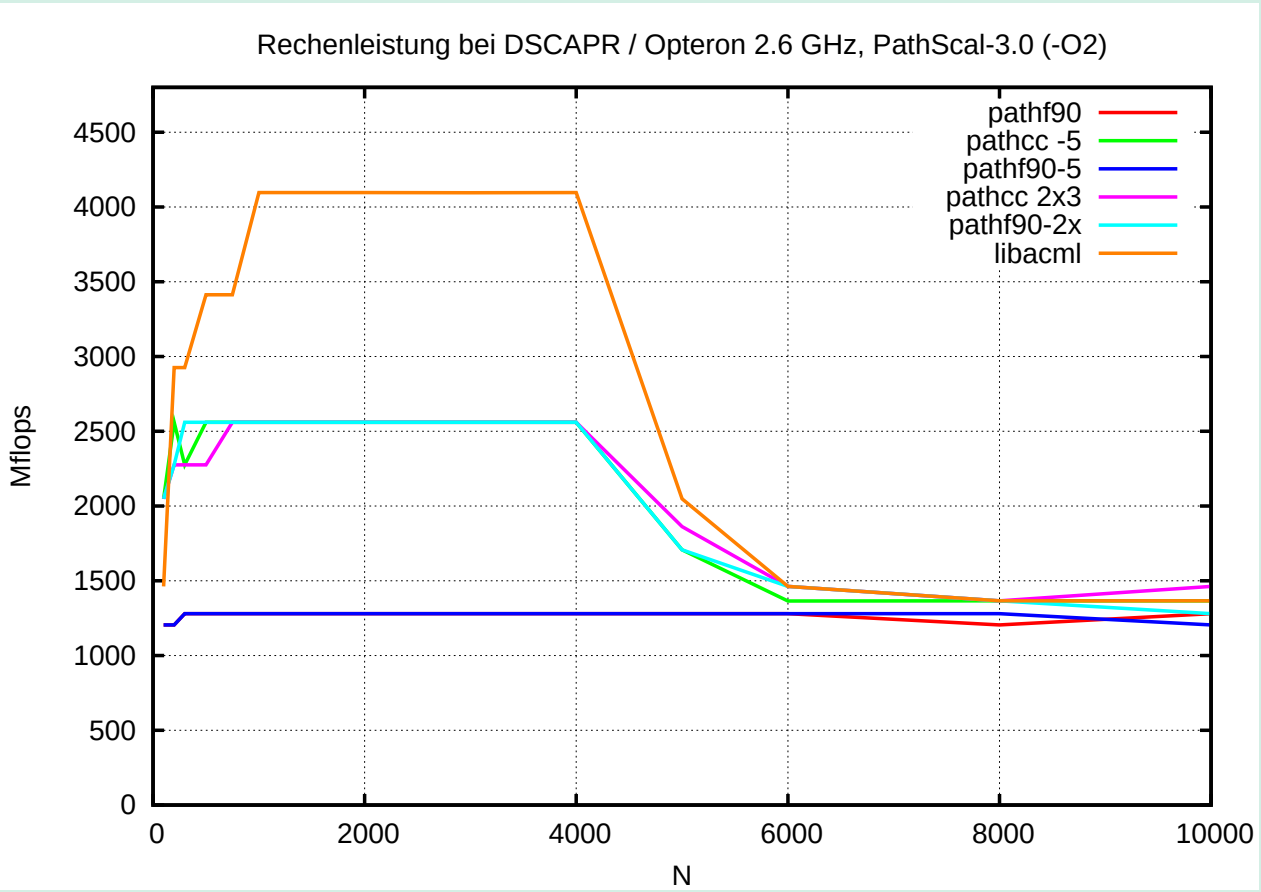




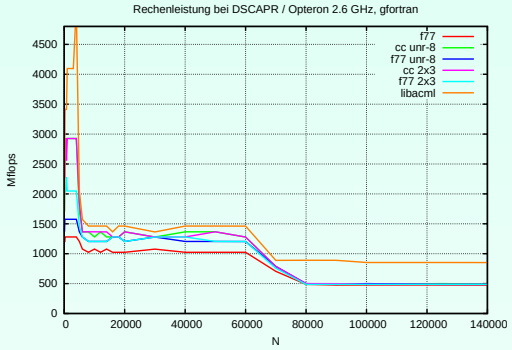


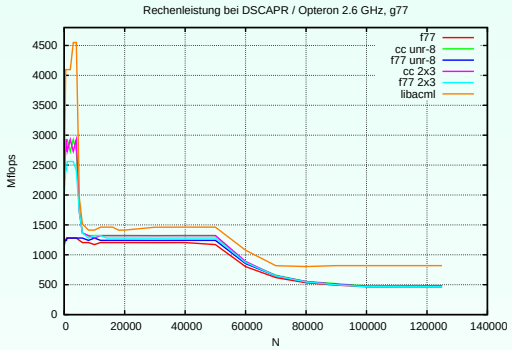
BLAS library: ACML = AMD Core Math Library

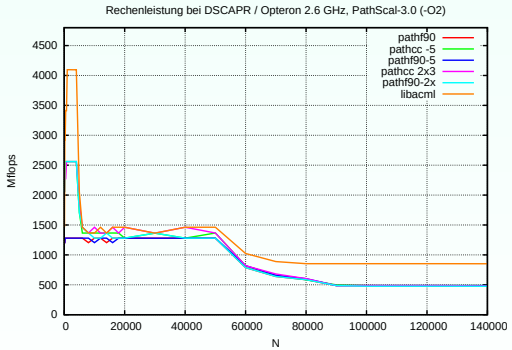
CHiC – PathScale-3.0 (-O2)



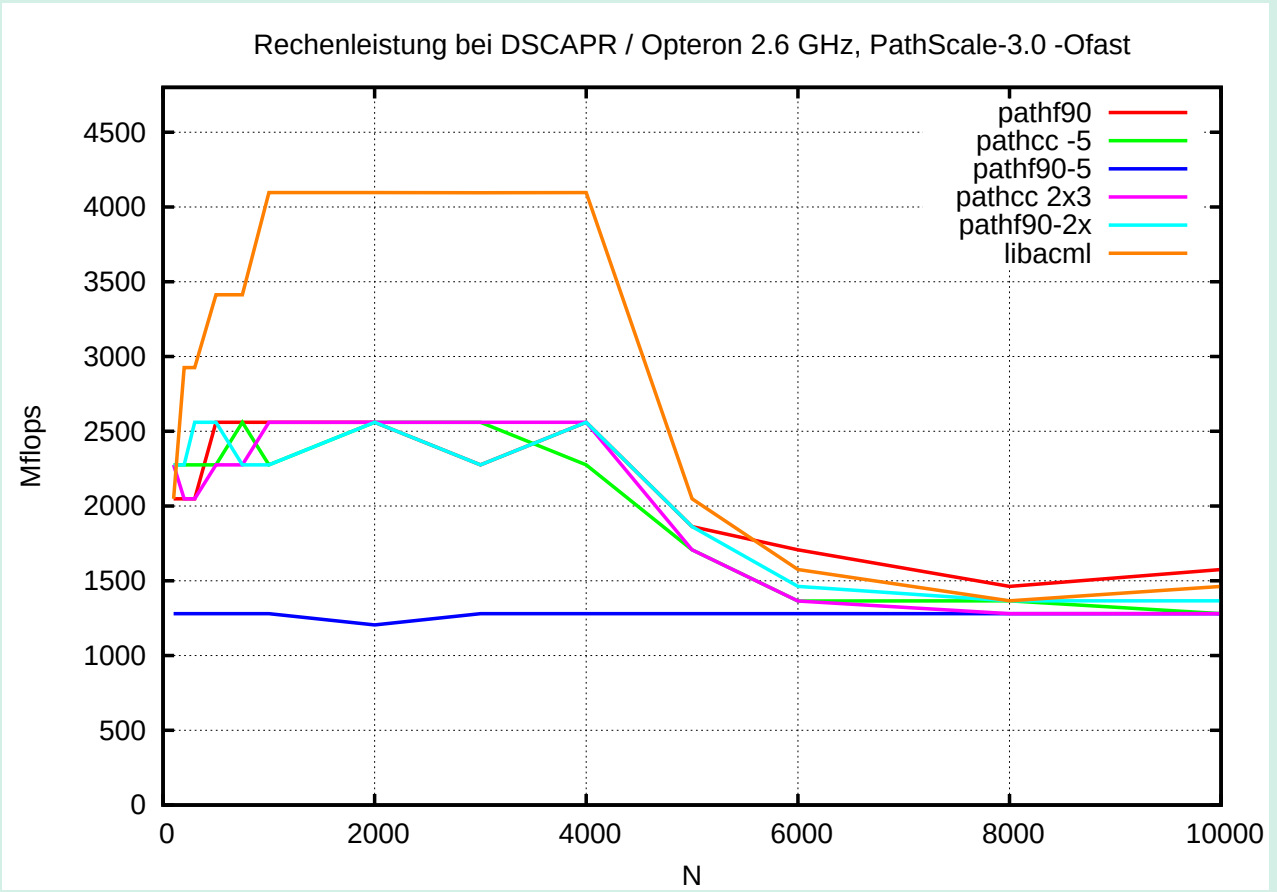
Single Node Performance on CHiC



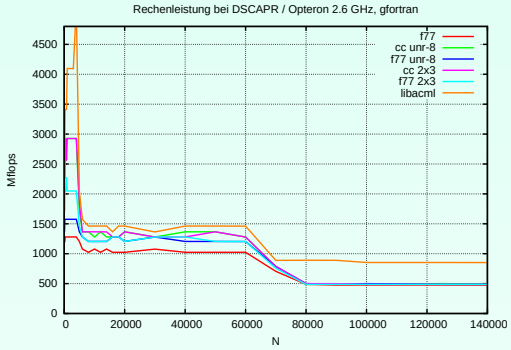


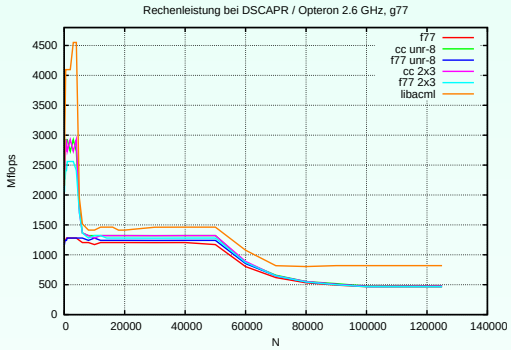


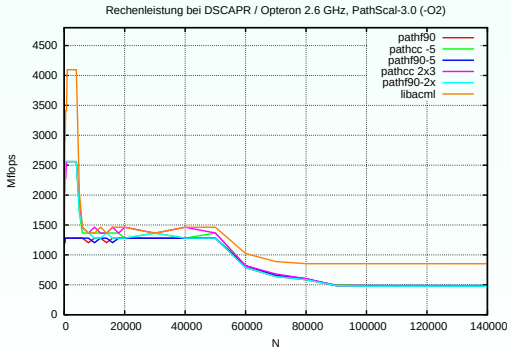
CHiC – PathScale-3.0 (-Ofast)



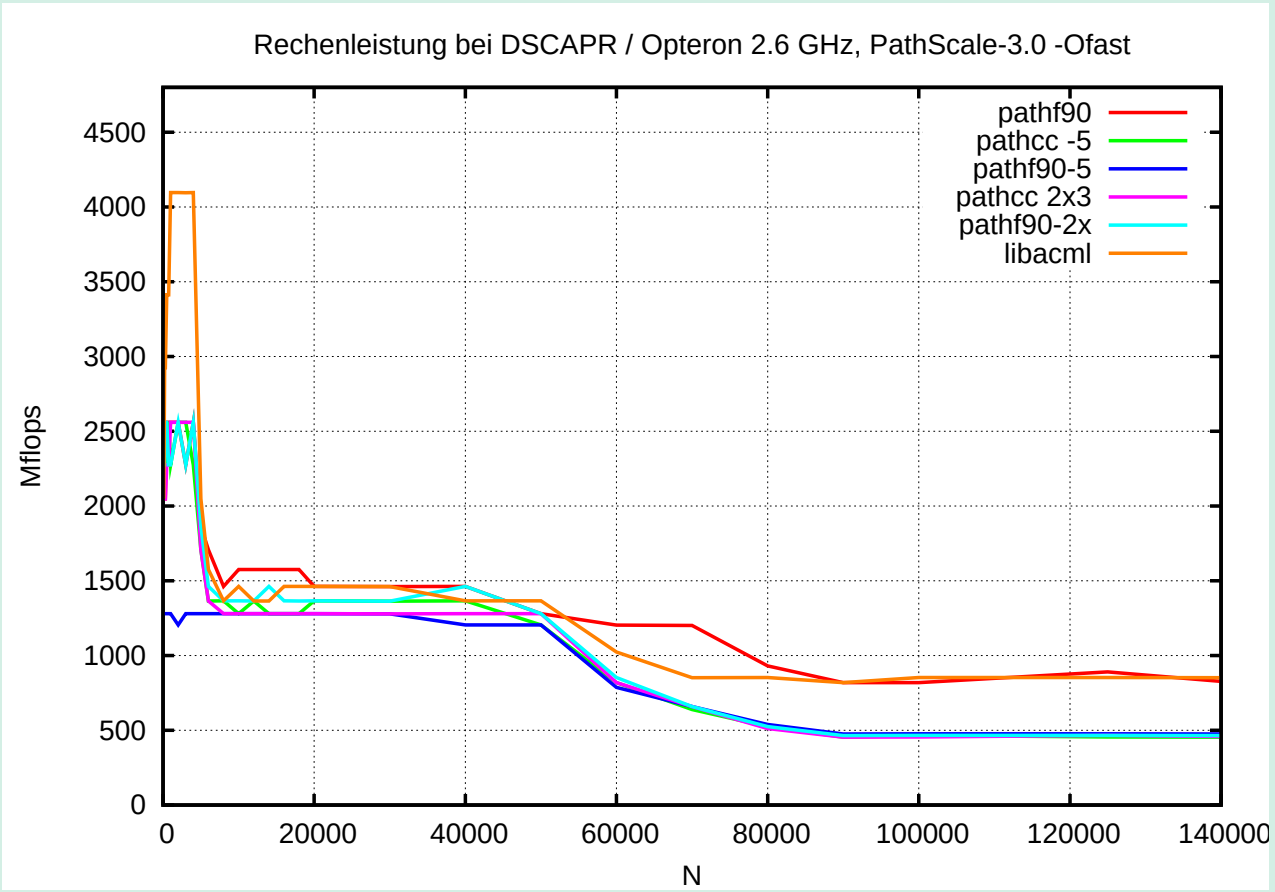
Single Node Performance on CHiC



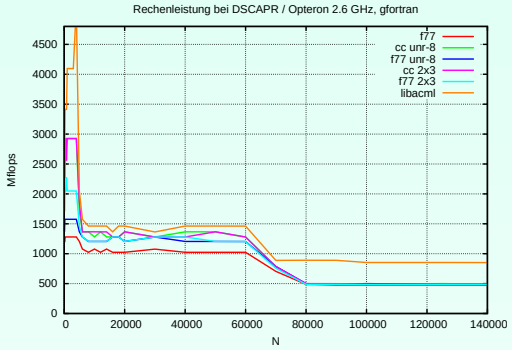


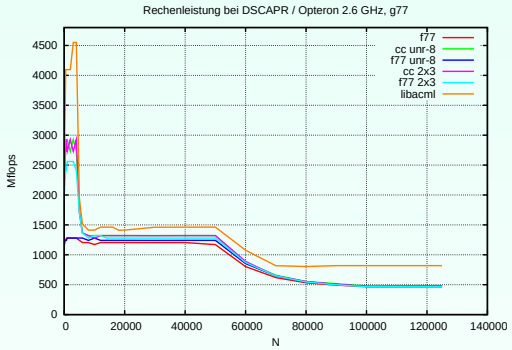


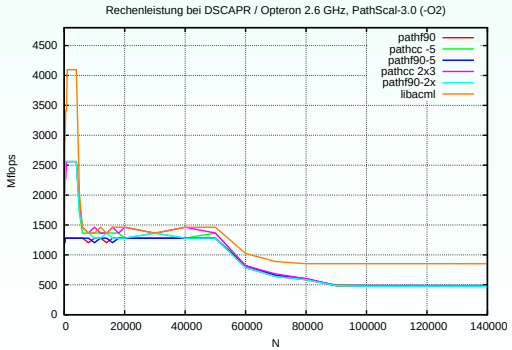
CHiC – PathScale-3.0 (-Ofast)



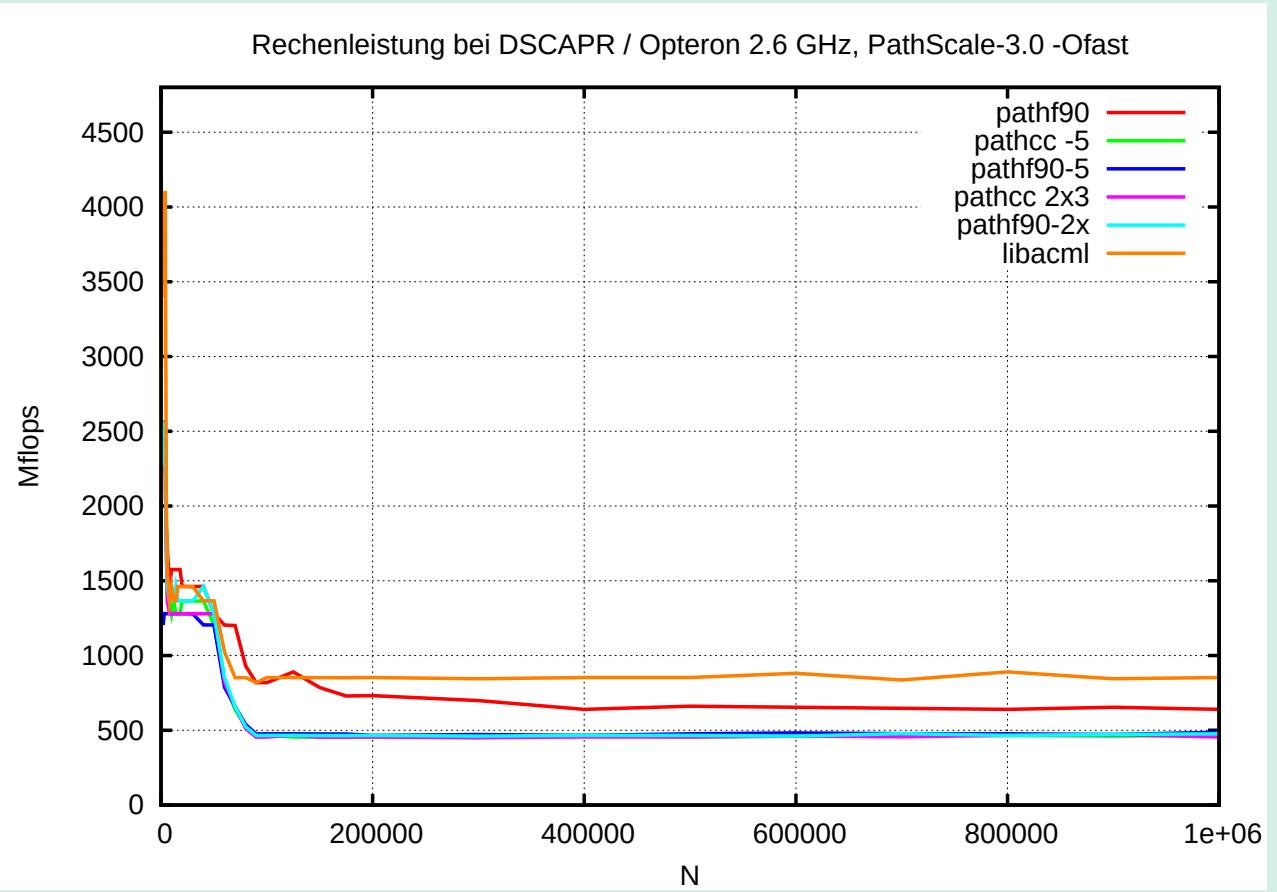
Single Node Performance on CHiC







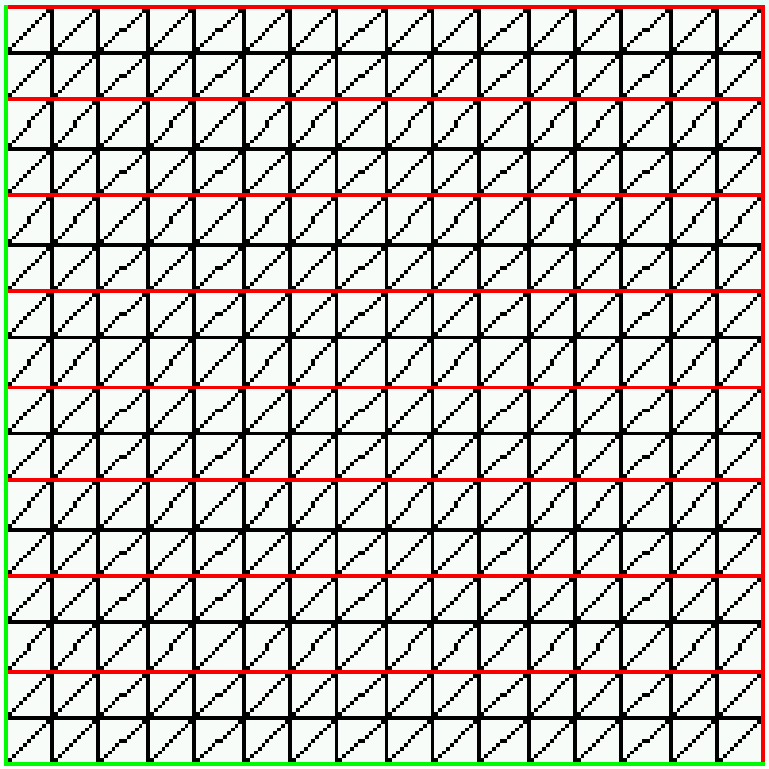
CHiC – PathScale-3.0 (-Ofast)

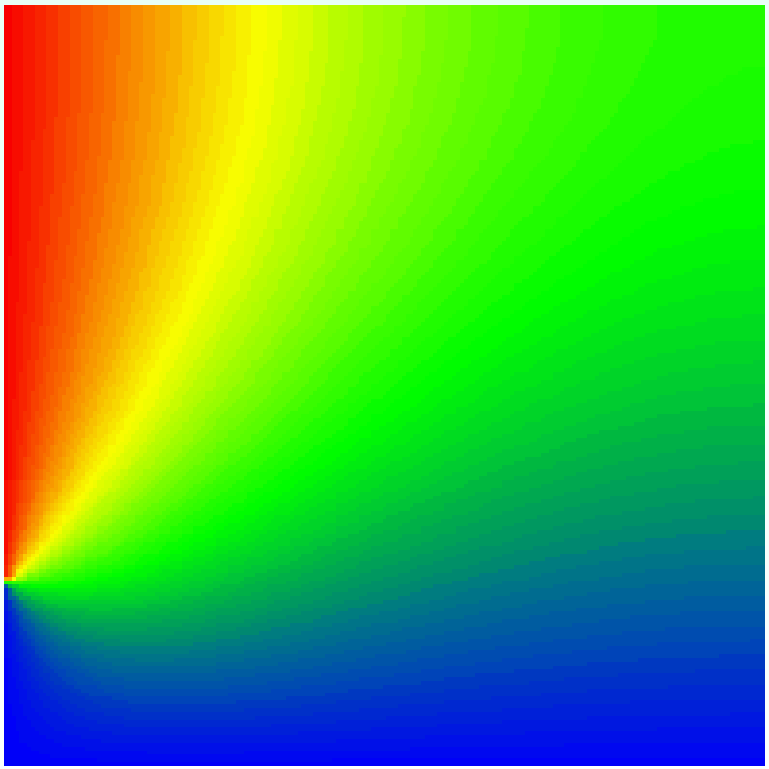


Single Node Performance

(2) Reference Example FEM-2D

Triangular mesh with 128 elements in coarse grid, previously used as reference example for many architectures





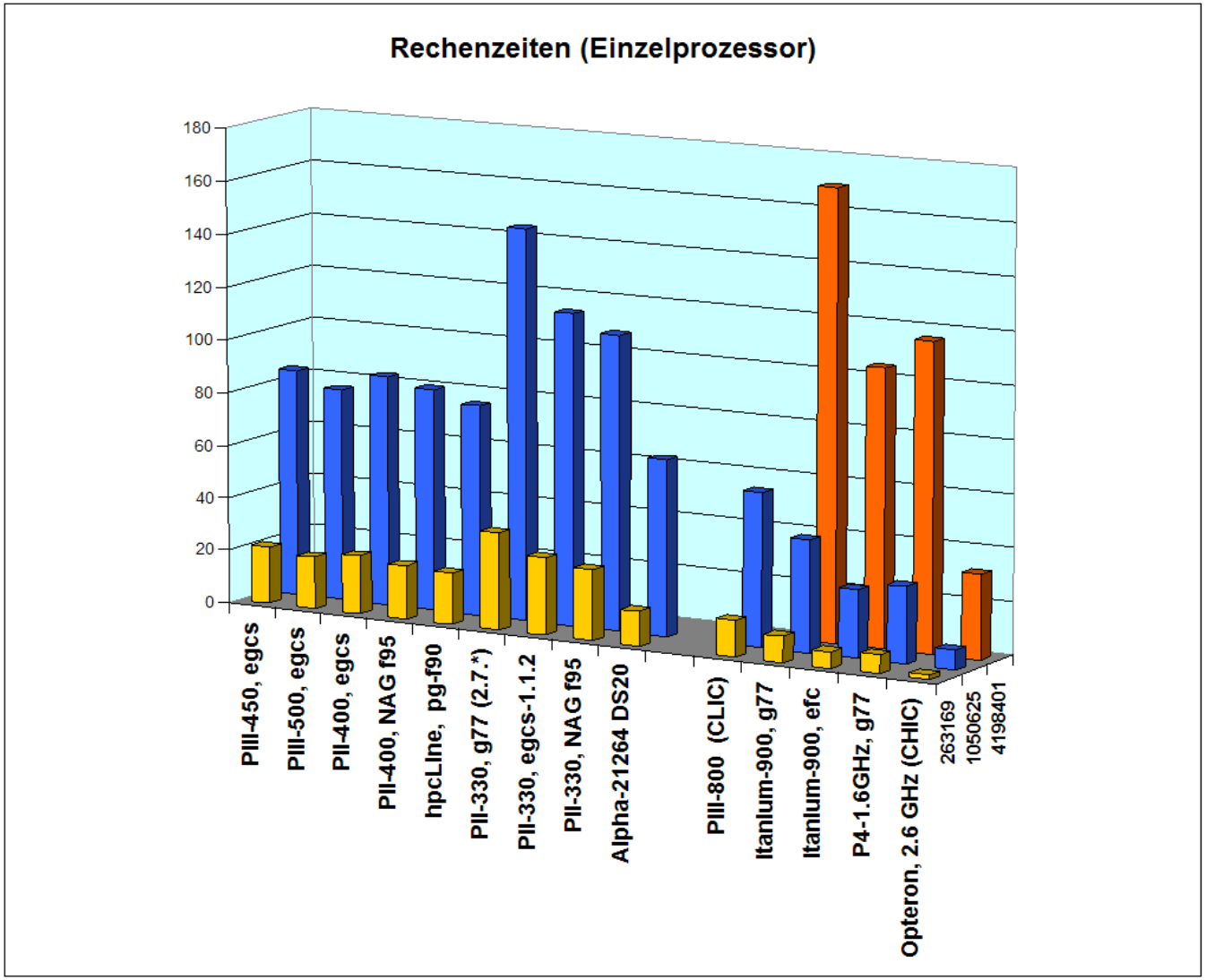
(2) Reference Example FEM-2D (1 Processor)

A selection of tested processors

(mostly before acquisition of CLiC, in 1999)
respectively 5-, 6-, 7-times uniformly refined mesh.

Refinement Level	5	6	7
Unknowns	263 169	1 050 625	4 198 401
#Iterations (PCG)	44	45	45
	Computing Time [s]		
hpcLine	19,3	79,3	—
Alpha 21264 DS20	13,0	66,2	—
PIII-800 (CLiC)	13,7	57,8	—
Itanium-900	6,1	25,5	104,4
P4 - 1.6 GHz	7,1	28,7	116,1
Opteron-2.6 GHz	1,7	7,3	31,8

(2) Reference Example FEM-2D (1 Processor)

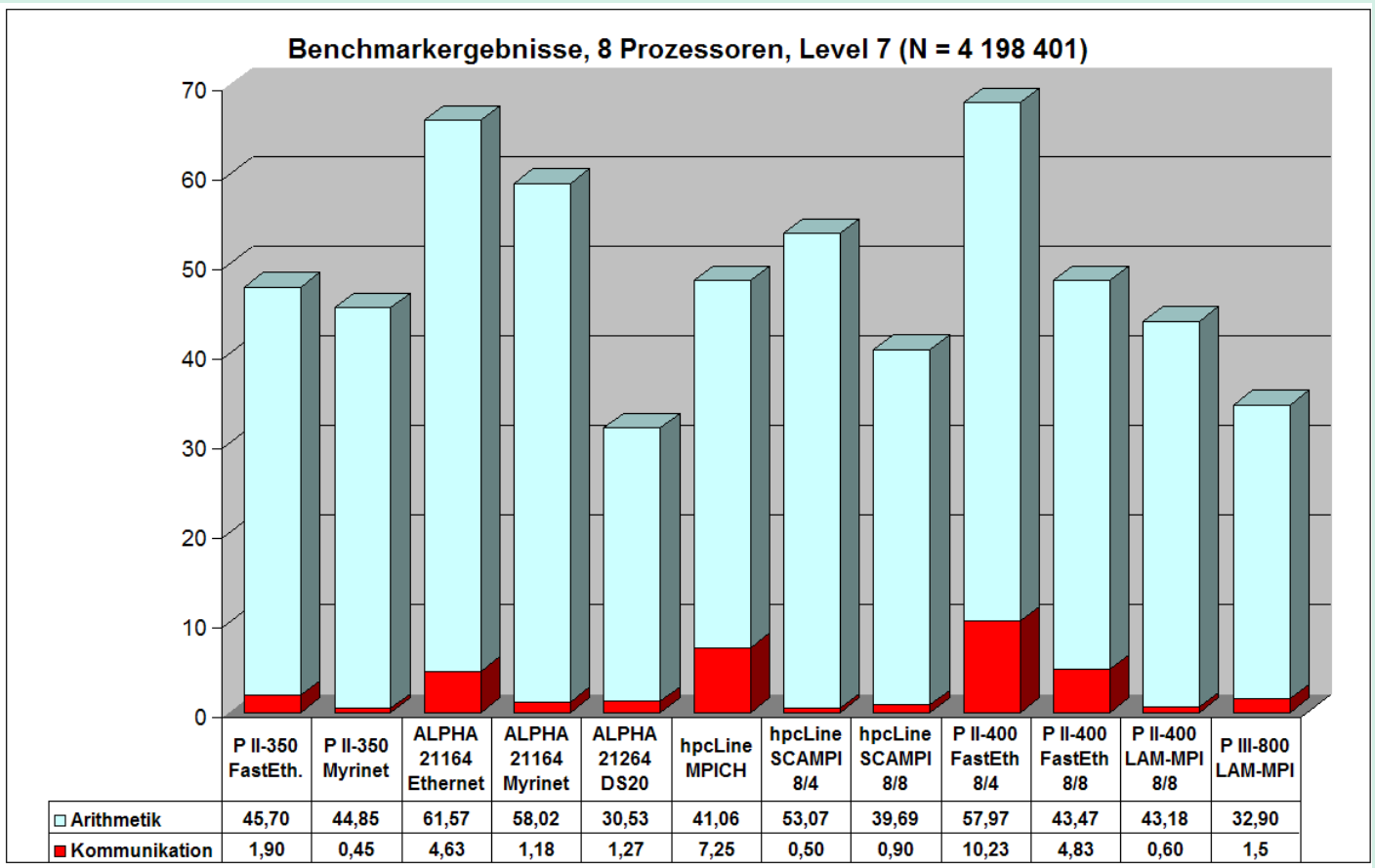


Parallel Performance (8-Processor-Cluster)

Total Computing Time

Example with 4 198 401 Unknowns (7-times refined mesh).

Clusters tested for acquisition of CLiC

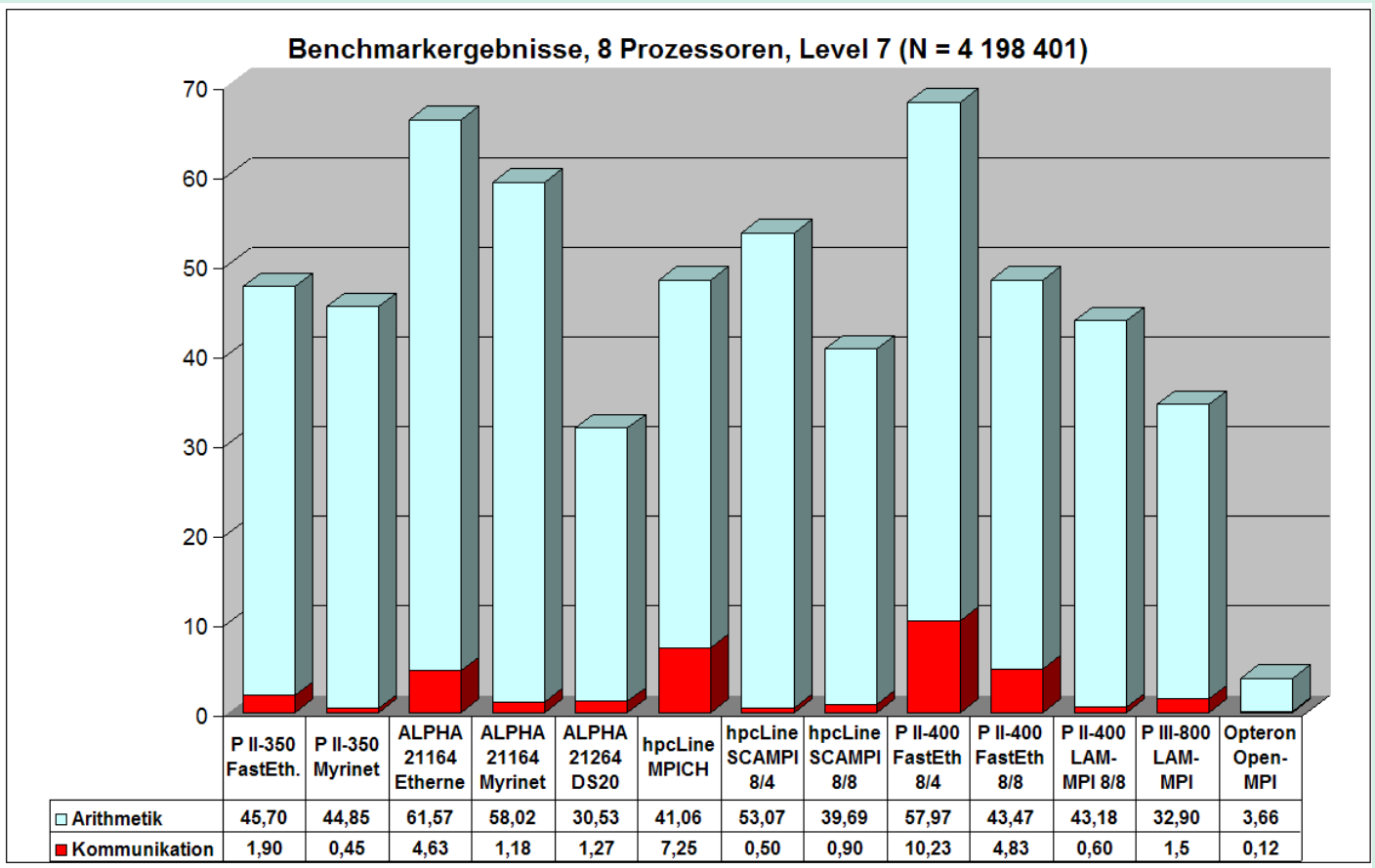


	P II-350 FastEth.	P II-350 Myrinet	ALPHA 21164 Ethernet	ALPHA 21164 Myrinet	ALPHA 21264 DS20	hpcLine MPICH	hpcLine SCAMPI 8/4	hpcLine SCAMPI 8/8	P II-400 FastEth 8/4	P II-400 FastEth 8/8	P II-400 LAM-MPI 8/8	P III-800 LAM-MPI
Arithmetik	45,70	44,85	61,57	58,02	30,53	41,06	53,07	39,69	57,97	43,47	43,18	32,90
Kommunikation	1,90	0,45	4,63	1,18	1,27	7,25	0,50	0,90	10,23	4,83	0,60	1,5

Parallel Performance (8-Processor-Cluster)

Total Computing Time

Example with 4 198 401 Unknowns (7-times refined mesh).
Clusters tested for acquisition of CLiC, compared with CHiC



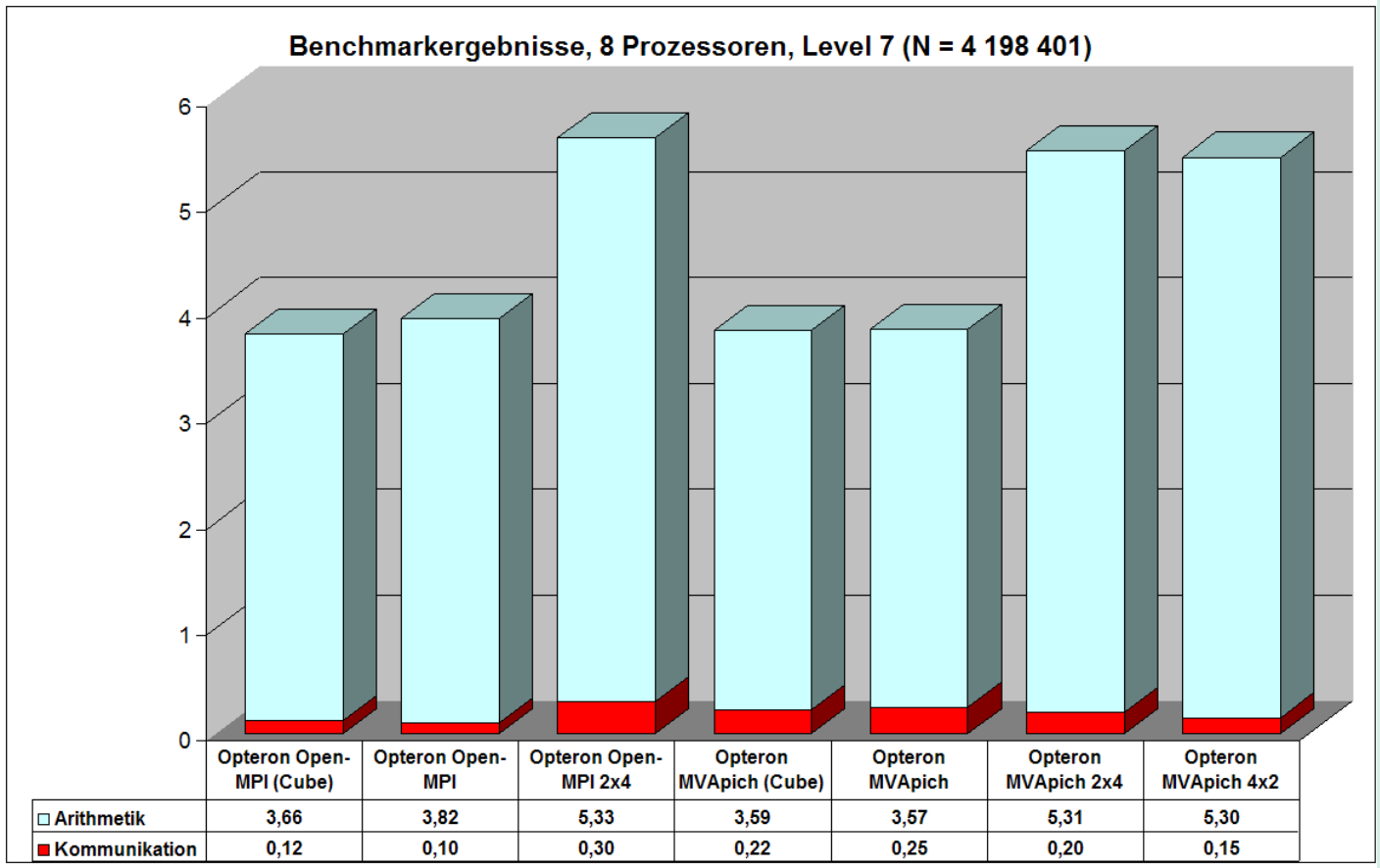
	P II-350 FastEth.	P II-350 Myrinet	ALPHA 21164 Etherne	ALPHA 21164 Myrinet	ALPHA 21264 DS20	hpcLine MPICH	hpcLine SCAMPI 8/4	hpcLine SCAMPI 8/8	P II-400 FastEth 8/4	P II-400 FastEth 8/8	P II-400 LAM-MPI 8/8	P III-800 LAM-MPI	Opteron Open-MPI
Arithmetik	45,70	44,85	61,57	58,02	30,53	41,06	53,07	39,69	57,97	43,47	43,18	32,90	3,66
Kommunikation	1,90	0,45	4,63	1,18	1,27	7,25	0,50	0,90	10,23	4,83	0,60	1,5	0,12

Parallel Performance (8-Processor-Cluster)

Total Computing Time

Example with 4 198 401 Unknowns (7-times refined mesh).

Different test situation on CHiC



	Opteron Open-MPI (Cube)	Opteron Open-MPI	Opteron Open-MPI 2x4	Opteron MVApich (Cube)	Opteron MVApich	Opteron MVApich 2x4	Opteron MVApich 4x2
Arithmetik	3,66	3,82	5,33	3,59	3,57	5,31	5,30
Kommunikation	0,12	0,10	0,30	0,22	0,25	0,20	0,15

Parallel Performance (64 and 128 procs.)

Lev.	Unknowns	#It[1]	64 proc. (64 × 1) Ass.	PCG	IO	64 proc. (16 × 4) Ass.	PCG	IO[2]
7	4 198 401	45	0,10	0,36	10%	0,11	0,49	10%
8	16 785 409	45	0,42	1,72	4%	0,44	2,68	5%
9	67 125 249	44	1,67	7,29	4%	1,75	11,34	5%
10	268 468 225	47	6,73	32,59	2%	7,00	49,84	5%

			128 Proc. (128 × 1)			128 Proc. (32 × 4)		
7	4 198 401	45	0,05	0,15	5%	0,06	0,2	30%
8	16 785 409	45	0,21	0,8	3%	0,22	1,3	6%
9	67 125 249	44	0,84	3,7	4%	0,89	5,6	5%
10	268 468 225	47	3,37	13,9	2%	3,52	23,1	5%
11	1 073 807 361	48	13,82	64,1	3%		—	

[1] Precond. CG, without coarse grid solver

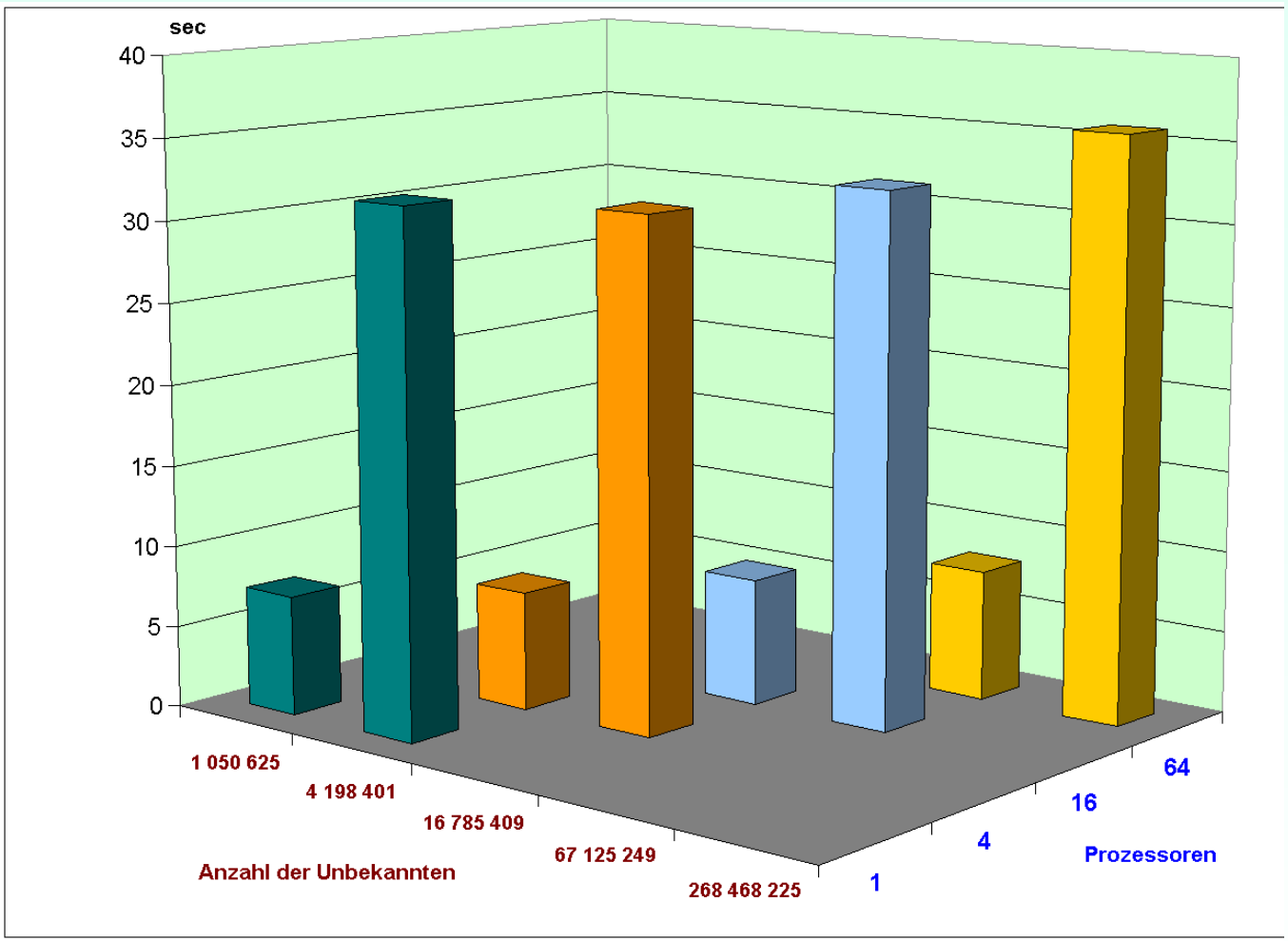
[2] Rough average among procs. (differing)

Scaling with the Problem Size

Each step of refinement: $4\times$ #unknowns with $4\times$ #processors

Scaling with the Problem Size

Each step of refinement: 4× #unknowns with 4× #processors



Performance of Global Communication

(1) Description of the Test:

- Which performance of the communication network can the user really get in his 'real-life' software environment?
- Therefore, the following is implemented in Fortran (the same as on CLiC, 5 years before):
  - Each processor: locally stored a (`double`) vector of length N.
  - Compute the global sum of all vectors over all processors.
  - The number of processors is $p = 2^n$.
- Two different implementations:
  - `Cube_DoD` (MPIcubecom)
    Hypercube-Routine based on `MPI_sendrecv`
  - `MPI_Allreduce` (MPIcom)
    should be the 'best' implementation by MPI

Performance of Global Communication

(1) Description of the Test:

- Which performance of the communication network can the user really get in his 'real-life' software environment?
- Therefore, the following is implemented in Fortran (the same as on CLiC, 5 years before):
  - Each processor: locally stored a (`double`) vector of length N.
  - Compute the global sum of all vectors over all processors.
  - The number of processors is $p = 2^n$.
- Two different implementations:
  - `Cube_DoD` (MPIcubecom)
    Hypercube-Routine based on `MPI_sendrecv`
  - `MPI_Allreduce` (MPIcom)
    should be the 'best' implementation by MPI

Performance of Global Communication

(2) Notes on evaluation:

- The `Cube_DoD`-version allows to determine the amount of data transferred from and to each processor, and from the measured time t a communication rate can be given in Mbit/s per processor or for the whole subcluster.
- Computation for $p = 2^n$ processors, vector length N:
  Packet length: $L = \frac{8N}{(1024)^2}$ [MByte],
  Total data flow: $G = n \cdot p \cdot L$, or per processor $2 \cdot n \cdot L$,
  Total rate: G/t [MByte/s],
  Rate per node: $8 \cdot (2 \cdot n \cdot L)/t$ [Mbit/s].
- Because MPI does not prescribe a certain way of data flow, the result of the `MPI_Allreduce`-version is more 'fictive'.
- For small packet lengths ($\approx$ 100 KByte) the measured time is too small for reliable results (other than on CLiC).

Performance of Global Communication

(2) Notes on evaluation:

- The `Cube_DoD`-version allows to determine the amount of data transferred from and to each processor, and from the measured time t a communication rate can be given in Mbit/s per processor or for the whole subcluster.
- Computation for $p = 2^n$ processors, vector length N:
  Packet length: $L = \frac{8N}{(1024)^2}$ [MByte],
  Total data flow: $G = n \cdot p \cdot L$, or per processor $2 \cdot n \cdot L$,
  Total rate: G/t [MByte/s],
  Rate per node: $8 \cdot (2 \cdot n \cdot L)/t$ [Mbit/s].
- Because MPI does not prescribe a certain way of data flow, the result of the `MPI_Allreduce`-version is more 'fictive'.
- For small packet lengths ($\approx$ 100 KByte) the measured time is too small for reliable results (other than on CLiC).

Performance of Global Communication

(3) Results obtained from measured times: Mb/s (each node!)

local vector length N	packets L [MB]	data flow G [GB]	Open-MPI (Cube)	Open-MPI (Reduce)	MVApich (Cube)	MVApich (Reduce)	CLiC
16 processors:							
2097152	16	1	9 309	1 862	10 240	10 240	141
8388608	64	4	9 525	1 837	11 703	10 180	142
32 processors:							
2097152	16	2,5	7 529	1 164	9 014	11 700	141
8388608	64	10	7 420	1 222	6 564	11 398	142
64 processors:							
2097152	16	6	8 533	753	5 485	3 938	141
8388608	64	24	7 062	752	5 535	3 990	141
128 processors:							
2097152	16	14	6 288	298	5 600	3 990	141
8388608	64	56	5 973	455	4 876	3 775	141

Total time was $\approx 0.1 \ldots 2$ s (for CLiC: $5 \ldots 50$ s)

Performance of Global Communication

(3) Results obtained from measured times: Mb/s (each node!)

local vector length N	packets L [MB]	data flow G [GB]	Open-MPI (Cube)	Open-MPI (Reduce)	MVApich (Cube)	MVApich (Reduce)	CLiC
16 processors:							
2097152	16	1	9 309	1 862	10 240	10 240	141
8388608	64	4	9 525	1 837	11 703	10 180	142
32 processors:							
2097152	16	2,5	7 529	1 164	9 014	11 700	141
8388608	64	10	7 420	1 222	6 564	11 398	142
64 processors (32×2):							
2097152	16	6	4 800	725	3 339	2 560	141
8388608	64	24	4 726	746	3 531	2 560	141
128 processors:							
2097152	16	14	6 288	298	5 600	3 990	141
8388608	64	56	5 973	455	4 876	3 775	141

Total time was $\approx 0.1 \ldots 2$ s (for CLiC: $5 \ldots 50$ s)

Summary

- Acceptable behavior of *cluster-friendly applications*;
  no significant differences in performance for various compilers and MPI installations
- Very good performance of the communication network fulfills our expectations.
  The percentage of communication is small, although the computing power has been massively increased.
- In comparison to single nodes, the dual-board-dual-core nodes show a reduction of *computing power* by upto 30% for our traditional parallel applications.

Summary

- Acceptable behavior of *cluster-friendly applications*;
  no significant differences in performance for various compilers and MPI installations
- Very good performance of the communication network fulfills our expectations.
  The percentage of communication is small, although the computing power has been massively increased.
- In comparison to single nodes, the dual-board-dual-core nodes show a reduction of *computing power* by upto 30% for our traditional parallel applications.

Summary

- Acceptable behavior of *cluster-friendly applications*;
  no significant differences in performance for various compilers and MPI installations
- Very good performance of the communication network fulfills our expectations.
  The percentage of communication is small, although the computing power has been massively increased.
- In comparison to single nodes, the dual-board-dual-core nodes show a reduction of *computing power* by upto 30% for our traditional parallel applications.

Optional Extras: Presentation in the Media

Media Reports Changing in Time

22.3./25.10.1994

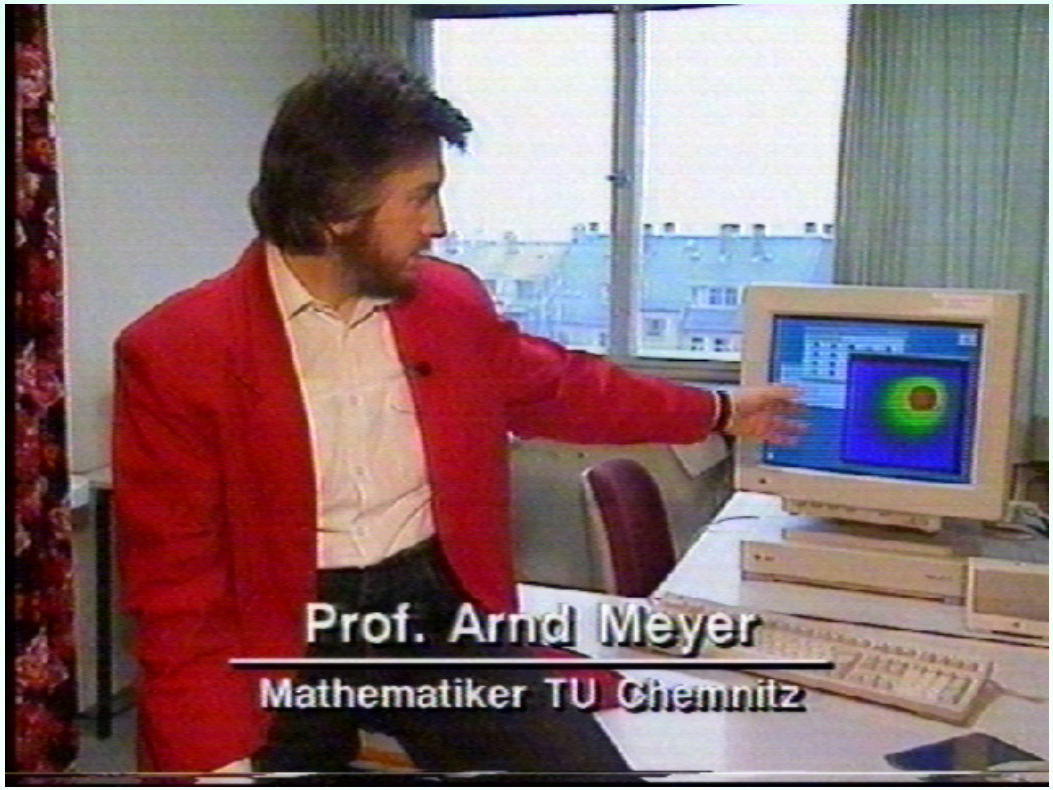
11.10.2000

07.02.2007

GCel-192 / GCPP-128 (1994)

return

GCel-192 / GCPP-128 (1994)



return

CLiC (2000)

return

CHiC (2007)

return

Optional Extras: Presentation in the Media

Media Reports Changing in Time

22.3./25.10.1994

11.10.2000

07.02.2007