Benchmarks on Current (Dual Core) CPUs

G. Wellein, G. Hager, T. Zeiser
RRZE

Application Performance

Turbulence simulation using the Lattice Boltzmann Method

Prof. Durst, Computational Fluid Dynamics FAU; P. Lammers, HLRS
Teraflop-Workbench
(Part of LRZ benchmark suite)

G. Wellein, P. Lammers, G. Hager, S. Donath, and Th. Zeiser, Proc. of ParCFD2005.
Towards Optimal Performance for Lattice Boltzmann Applications on Terascale Computers

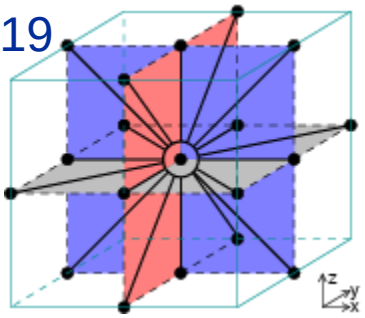
G. Wellein, T. Zeiser, G. Hager and S. Donath, to appear in Computer&Fluids.
On the Single Processor Performance of Simple Lattice Boltzmann Kernels

Th. Pohl, F. Deserno, N. Thürey, U. Rüde, P. Lammers, G. Wellein, and T. Zeiser, in Proc. Supercomputing 2004, Pittsburgh, PA, 6.-12. Nov. 2004.
Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures

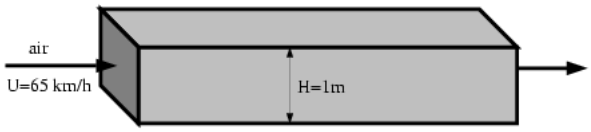
TFlop/s Computing with LBM
Introduction

- Performance evaluation/optimization of LBM application (BEST):
  - Fortran90
  - Push-Method (Collide-Stream)
  - Two Grids
  - MPI parallelisation
  - Variations:
    - Compressed Grid – no performance gain
    - 1D-/3D-Blocking – no performance gain
    - Temporal Blocking
  - A serial kernel was extracted for performance optimization

D3Q19



- Parallel benchmark runs were done for turbulent channel flow



TFlop/s Computing with LBM
Code Structure

```
double precision f(0:xMax+1,0:yMax+1,0:zMax+1,0:18,0:1)
do z=1,zMax
   do y=1,yMax
      do x=1,xMax
         if( fluidcell(x,y,z) ) then
            LOAD f(x,y,z, 0:18,t)
            Relaxation (complex computations)
            SAVE f(x  ,y  ,z  , 0,t+1)
            SAVE f(x+1,y+1,z  , 1,t+1)
            SAVE f(x  ,y+1,z  , 2,t+1)
            SAVE f(x-1,y+1,z  , 3,t+1)
            …
            SAVE f(x  ,y-1,z-1,18,t+1)
         endif
      enddo
   enddo
enddo
```

Collide

Stream

Split up loop on IA32/IA64 CPUs

Optional data layout

f(0:18, 0:xMax+1, 0:yMax+1, 0:zMax+1, 0:1)

is not optimal for cache based architectures

TFlop/s Computing with LBM
Performance – Basics

- Performance unit: Mega Lattice Site Updates per Second (MLUPS)
- Per Lattice Site update we need:
  - ~ 170-250 Floating Point Ops. -> 5 MLUPS ~ 1 GFlop/s
  - ~ 40 (+20 write allocate) 8 byte transfers between CPU & Memory
- Estimate max. performance (200 Flops per Lattice Site):

	Peak Perf. [GFlop/s]	Max. MLUPS	Bandwidth [GByte/s]	Max. MLUPS	**Measure 128^3**
Intel Xeon	6.8	34	5.3	**11.8**	**5.1** (43%)
AMD Opteron	4.4	22	6.4	**14.0**	**5.1** (36%)
Intel Itanium2	5.6	28	6.4	**14.0**	**8.5** (61%)
CRAY X1	12.8	**64**	34.1	112	**34.9** (55 %)
NEC SX6+	9.0	**45**	36.0	118	**41.3** (92 %)

- Vector performance not limited by memory bandwidth!
- There is room for improvement on IA32 compatible & CRAY X1

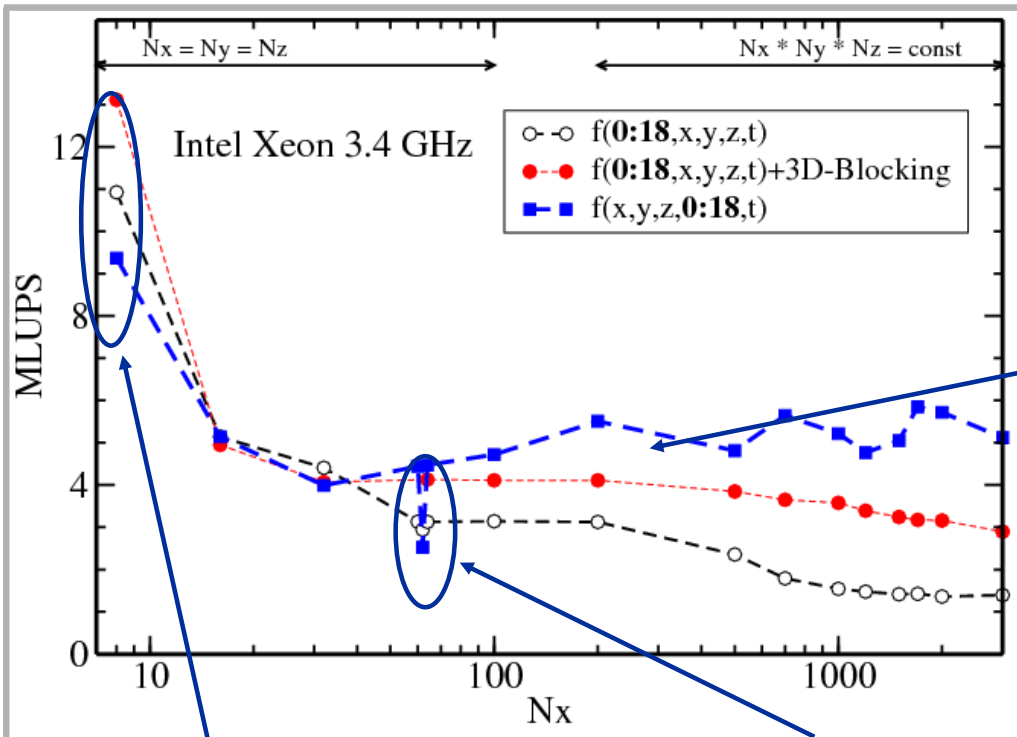
LBM
Code tuning – Splitting innermost x-loop

- Splitting the innermost x loops in 3 separate x-loops
  - Increases number of floating point ops per lattice site (~250 Flop)
  - Increases Data transfer Cache – CPU (intermediate results)
  - Reduces the pressure on the write combine buffers on IA32
  - Simplifies Instruction Scheduling for IA64

		Original	**Tuned**
Intel Xeon/Nocona	3.4 GHz	4.5 MLUPs	**4.9 MLUPs**
AMD Opteron848	2.2 GHz	2.9 MLUPs	**4.7 MLUPs**
AMD Opteron272*	2.2 GHz	3.6 MLUPs	**5.8 MLUPs**
PentiumD*	2.8 GHz	3.0 MLUPs	**4.3 MLUPs**
Intel Itanium2	1.4 GHz/1.5 MB L3	7.9 MLUPs	**8.5 MLUPs**

* Dual Core Systems

LBM
Data layout



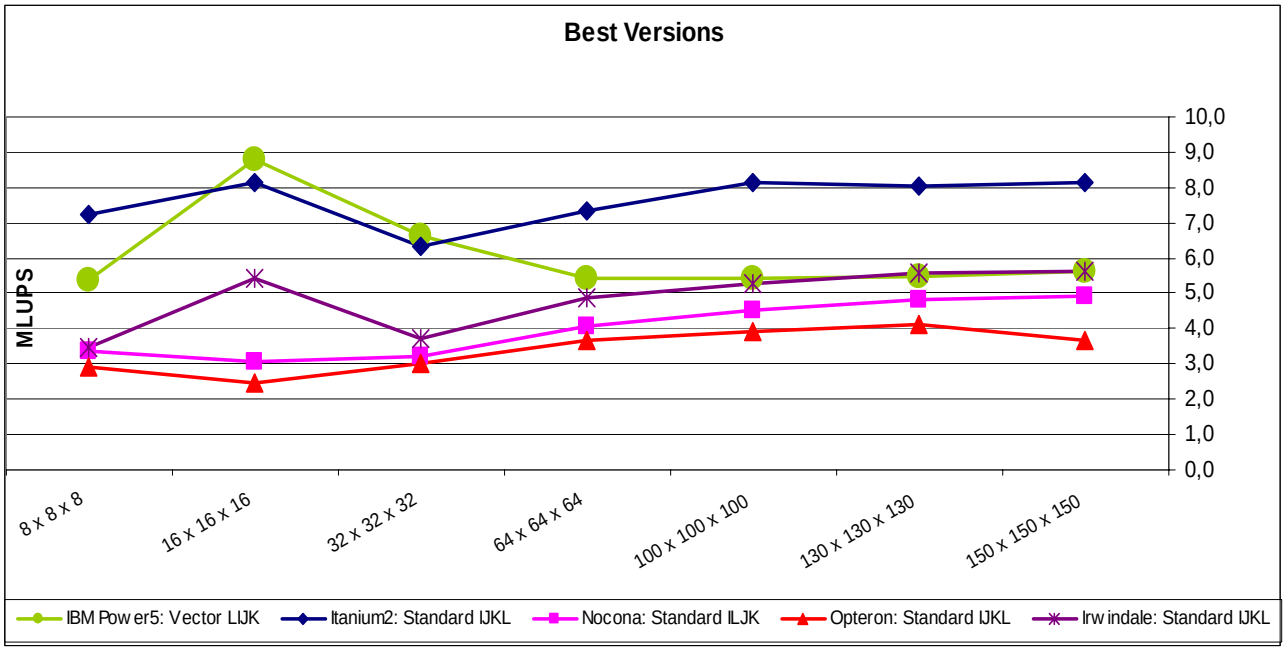
L2 cache hit ratio ~ 94%

Optimal use of data loaded from main memory

Full data set fits to L2 cache

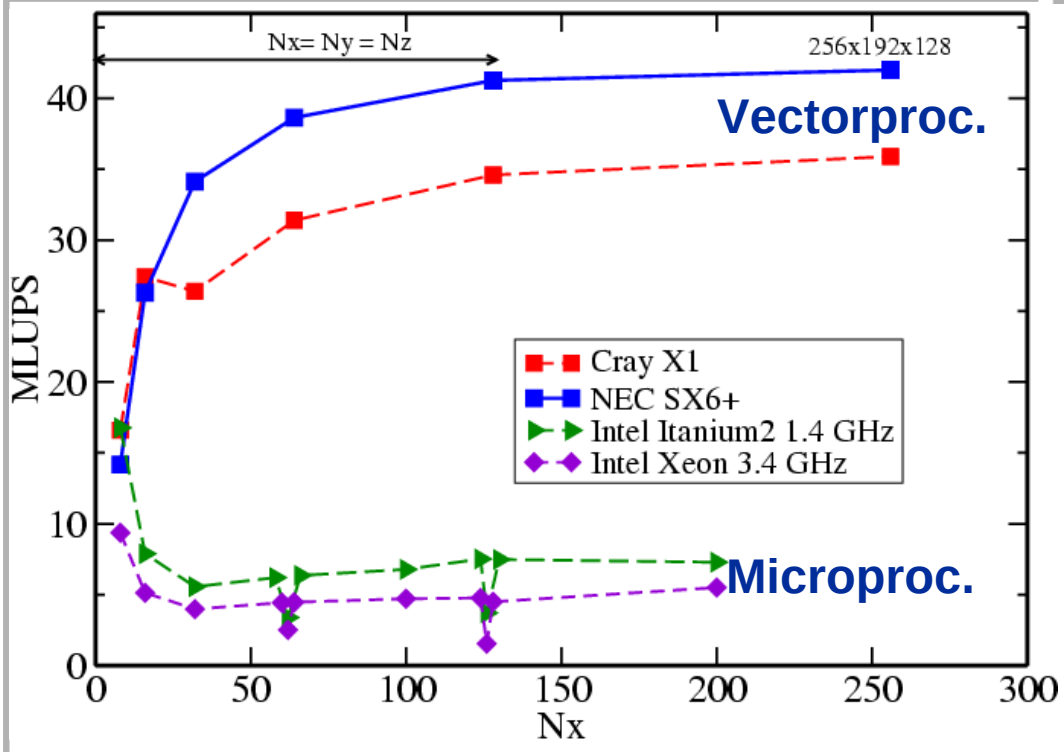
Be aware of cache trashing effects: Nx=2^n

LBM
Data layout & implementation: Power4/5 ???



- (I,J,K,0:18) best performance on all systems – except IBM Power5
- Manual fusing of the 3 spatial loops is essential on IBM Power5

LBM
Vector processor characteristics



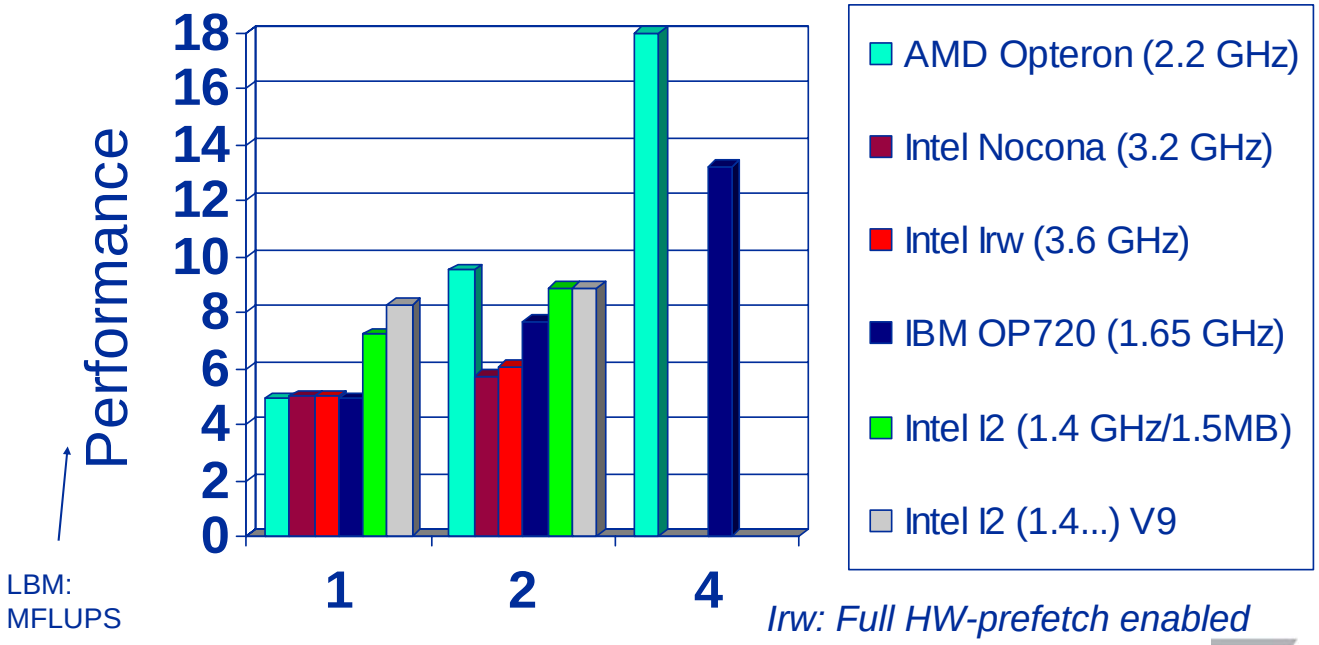
Vector version:

Fuse 3 spatial loops to long vector loop!

CRAY X1: Disabled vector caches / room for improvement

LBM
Scalability of SMP/ccNUMA nodes (OpenMP)

- Intel EM64T Compiler 8.1.024 (-O3 –xW): Intel & AMD
- Intel Xeon/Irwindale (3.6 GHz; 2 MB L2)
- IBM: Open Power720 (Power5): 4-fway; xlf90_r –qhot –O5 –qsmp=omp



Irw: Full HW-prefetch enabled

LBM
Scalability of Dual-Core CPUs (OpenMP)

- Intel PentiumD: 1 socket; 2 cores (2.8 GHz, 1 MB L2)
- AMD Opteron272: 2 sockets; 2 cores (2.2 GHz; 1 MB L2) (SUN V20Z)
- AMD Opteron875: 4 sockets; 2 cores (2.2 GHz; 1 MB L2) (transtec)
- OpenPower 720: 2 sockets; 2 cores (IBM Power5 1.65 GHz)

(Performance in MLUPs)

	1 socket		2 sockets		4 sockets	
threads/ socket	1	2	1	2	1	2
PentiumD	4.3	**6.3**	---	---	---	---
Opteron275	5.7	7.7	11.4	**15.1**	---	---
Opteron875	5.6	7.3	10.9	14.5	21.1	**27.9**
OPower720	n.a.	n.a.	n.a.	**13.2**	---	---

Speed-Up of 2nd core ~ 30%

LBM
Scalability of Dual-Core / SMP systems (OpenMP)

- AMD Opteron275: 2 sockets; 2 cores (2.2 GHz; 1 MB L2) (SUN V20Z)
- OpenPower 720: 2 sockets; 2 cores (IBM Power5 1.65 GHz)

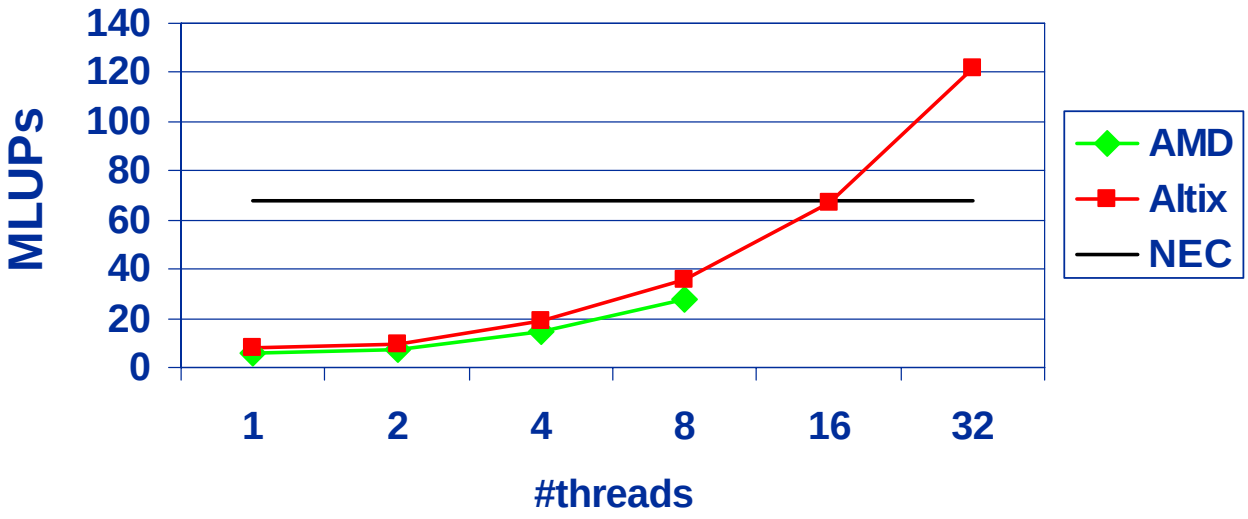
(Performance in MLUPs)

	Clock	Bandwidth	1 thread	2 threads	4 threads
Intel Prestonia	2.66 GHz	**4.2 GB/s**	2.6	**3.0**	---
Intel Nocona	3.4 GHz	**5.3 GB/s**	4.8	**5.5**	---
Intel Irwindale	3.6 GHz	**5.3 GB/s**	5.0	**6.0**	---
PentiumD	2.8 GHz	**6.4 GB/s**	4.3	**6.3**	---
Opteron275	2.2 GHz	**12.8 GB/s**	5.7	7.7	**15.1**
OPower720	1.65 GHz	**~20 GB/s**	n.a.	n.a.	**13.2**

LBM
ccNUMA vs. Vector-CPU

- AMD Opteron875: 4 sockets; 2 cores (2.2 GHz; 1 MB L2) (transtec)
- SGI Altix3700 Bx2: 32 CPUs (Itanium2; 1.6 GHz; 6 MB L3)
- NEC SX8 1 CPU (2 GHz)

Strong Scaling (128^3)



LBM
Remote Memory access via ccNUMA

- Place data and compute thread on different nodes to measure the potential of the ccNUMA interconnect

  `taskset -c i numactl -m k ./lbmkernel`

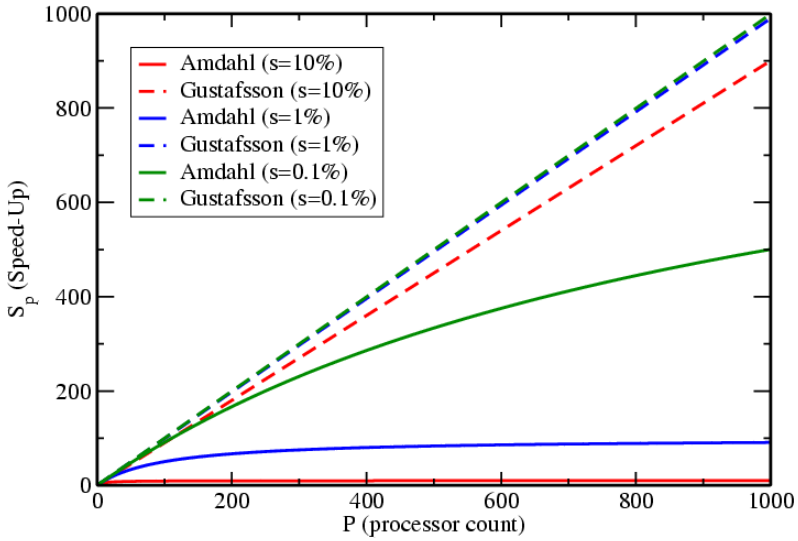
  i=0,…,N_{CPU}-1 ; k=0,…,(N_{CPU}/2) -1
- AMD Opteron275 (SUN V20z): HT-1000 (4 GB/s per direction)
- AMD Opteron848 (4-way)
- SGI Altix3700 Bx2 (32-way): NUMALink4 (3.2 GB/s per direction)

	-m 0	-m 1	-m 2	-m 3
AMD 275	**5.8**	**4.6**		
AMD 848	**4.7**	**4.0**	**4.0**	**3.5**
Altix	7.9	6.0	5.5 (-m 7)	5.5 (-m 15)

TFlop/s Computing with LBM
Parallel Performance - Basics

- **Strong Scaling: Total problem size (e.g. N^3) is fixed. Amdahl`s law limits speed-up (s: Serial fraction; P: processors)**

$$S_P = \frac{P}{(1+ s\,(P-1))}$$



Remember:

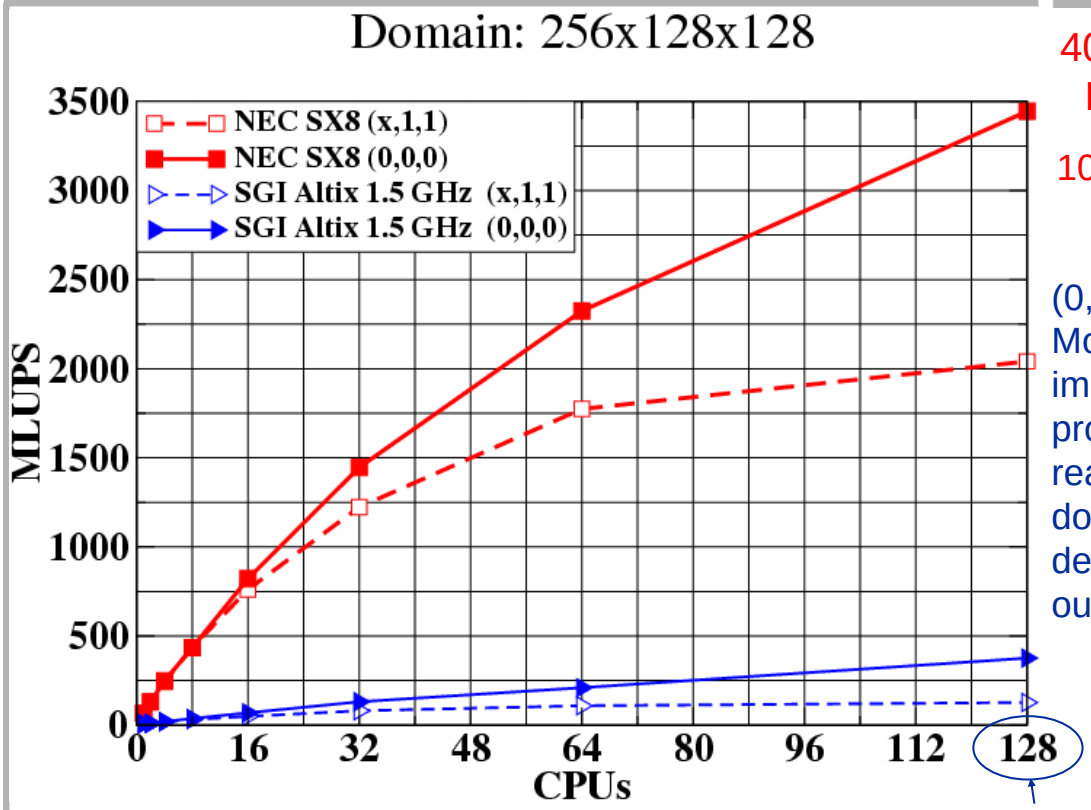
$T_{Comp} \sim N^3 / P$

$T_{Comm} \sim N^2 / P^{2/3}$

- **Weak Scaling: Total problem size (e.g. N^3) increases linearly with processor count (Gustafsson`s law)**

$S_P = s + (1-s) * P$

TFlop/s Computing with LBM:
Strong Scaling



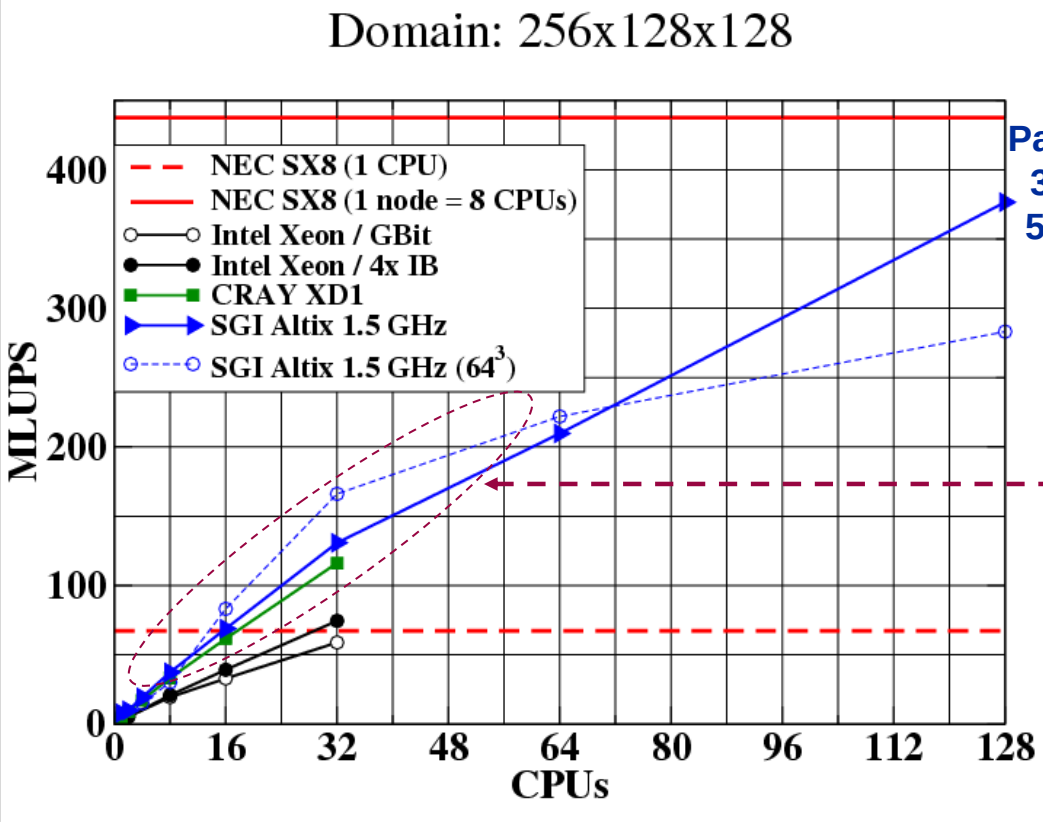
40% parallel Efficiency

1000 time steps in 1 second

(0,0,0): Most MPI implementations provide a reasonable domain decomposition for our channel

32.000 Lattice Sites per CPU

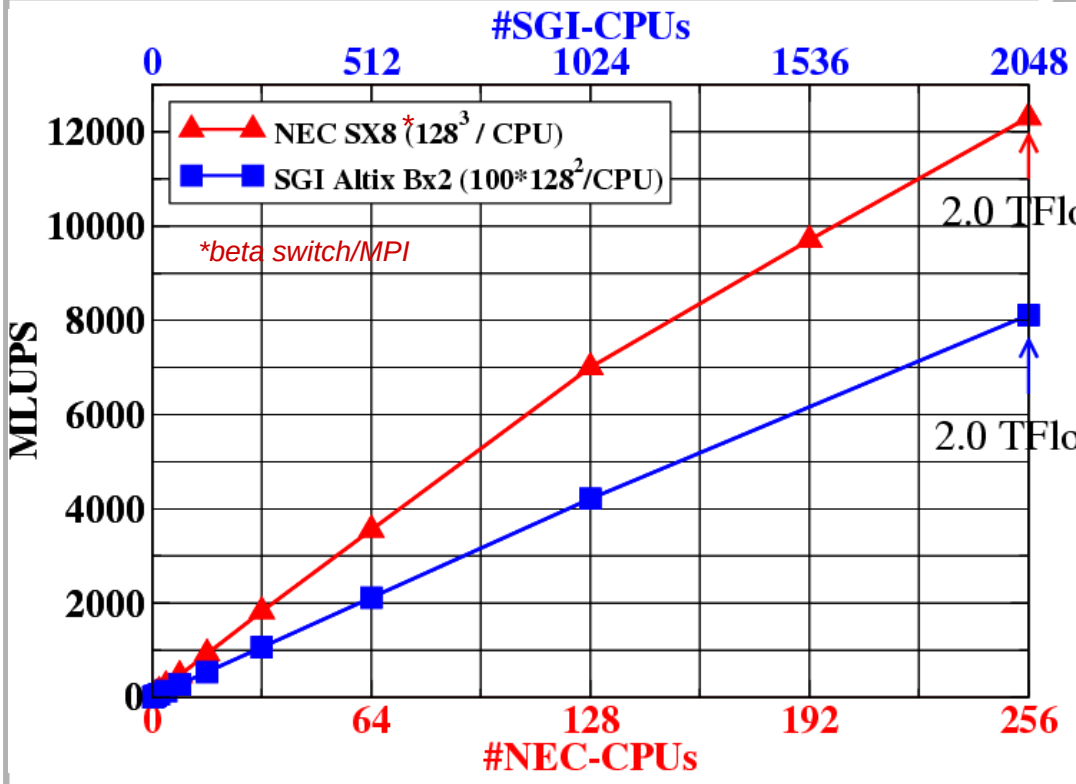
TFlop/s Computing with LBM: Strong Scaling



Parallel Efficiency:
37% (CPU-base)
59% (node-base)

Performance increases super-linear: Aggregate Caches Size > Size of data set (~80 MB)!

TFlop/s Computing with LBM: Weak Scaling



$0.5*10^9$ Lattice Sites

2.0 TFlop/s

$3.3*10^9$ Lattice Sites

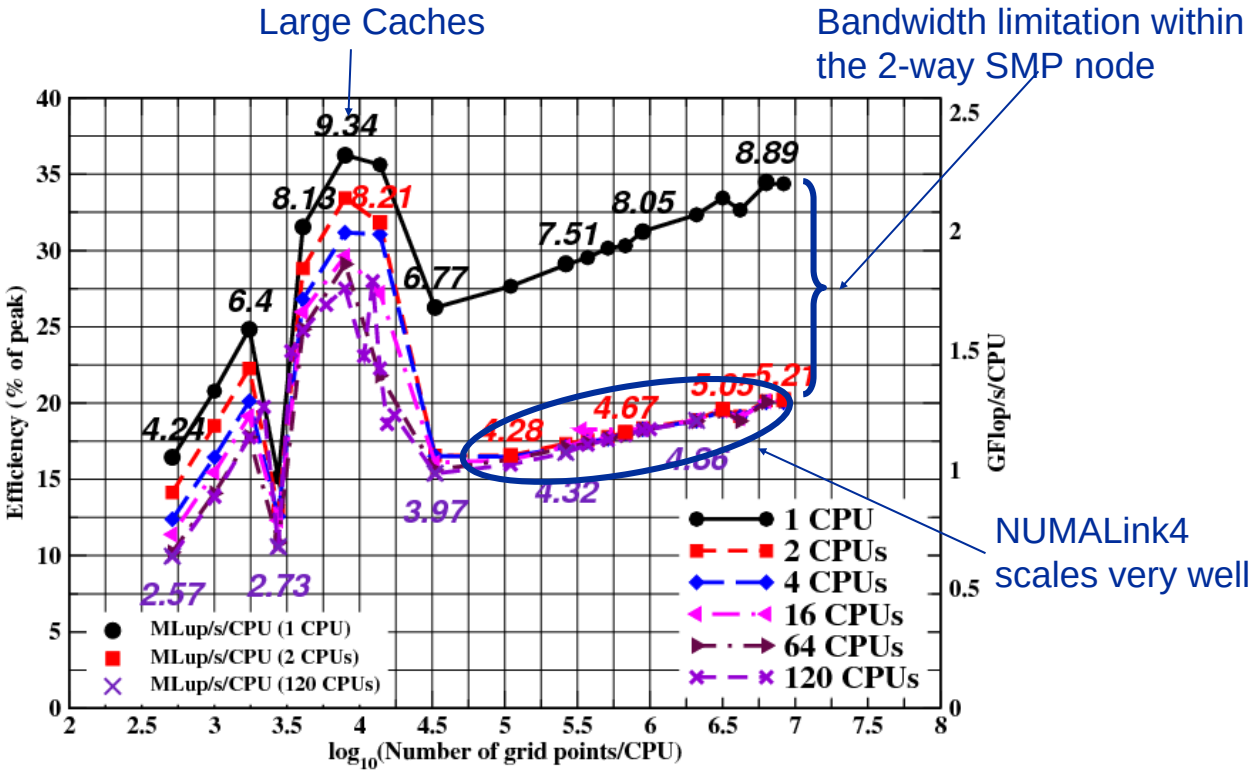
2.0 TFlop/s

FLOP/LUP ~

- 170 (NEC)
- 250 (SGI)

Projection for full HLRS system (576 CPUs) > 5 TFlop/s
($6*10^9$ Lattice Sites on 256 CPUs: 14.8 GLUPS ~ 2.6 TFlop/s)

LBM scaling
SGI Altix 3700 Bx2



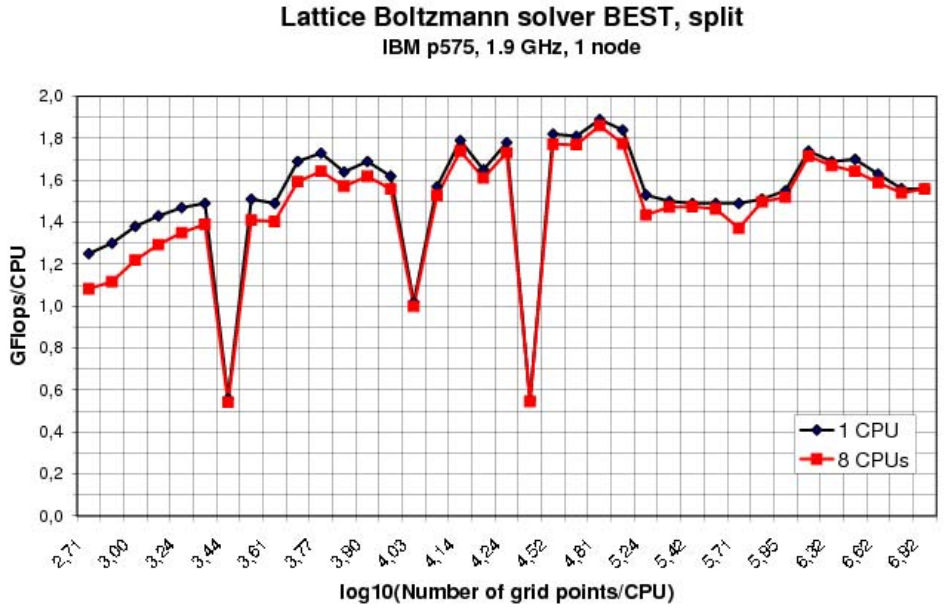
New LRZ system >1.25 GFlop/s per CPU -> Total performance > 6 TFlop/s

LBM scaling
1 node IBM p575 (8 CPUs)

Plenty of bandwidth within the node – Scales perfectly

1 node (8 CPUs) ~ 13 GFlop/s ~ 10 SGI Altix CPUs

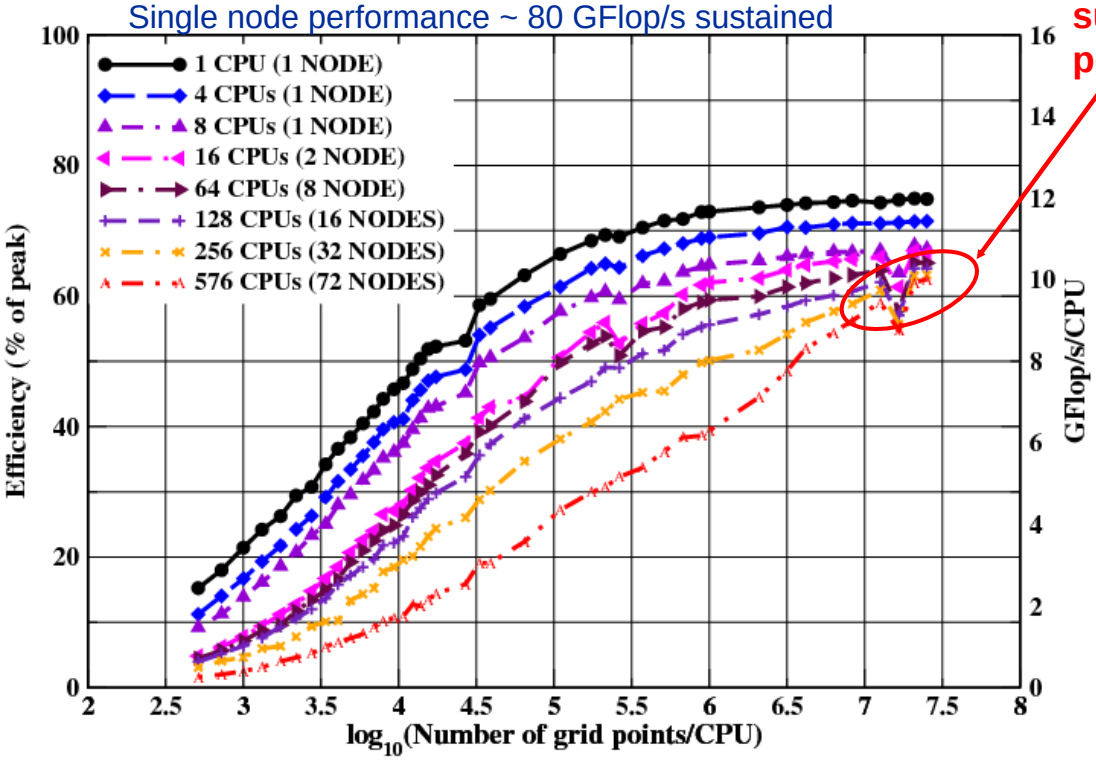
Large gap between theoretical and sustainable bandwidth on IBM systems…



LBM scaling
NEC SX8 (1 node = 8 CPUs)

Speed-Up within 1 node ~7-7.5

Single node performance ~ 80 GFlop/s sustained



1–5 TFlop/s sustained performance

TFlop/s Computing with LBM:
Weak Scaling: Vector vs. the rest of the world ?!

Vector systems are still a class of their own for LBM codes

Full HLRS system (576 SX8-CPUs): 32.000 MLUPS (5.5 TFlop/s)

Assuming weak scaling, this is equivalent to

- **Cluster of ~ 7.000 AMD Opteron** **[12:1]**
- **~14.000 Intel Xeon CPUs** **[24:1]**
- **SGI Altix with ~6.500 Intel Itanium2 CPUs** **[11:1]**
- **IBM BG/L with ~ 32.000 CPUs** (cf. next slide) **[60:1]**

So far "microprocessor/-friendly" test case has been considered (empty channel):

- **No indirect addressing**
- **Contiguous memory access**

LBM application performance: Price Performance

	CPUs	MLUPS	Performance	Price (K€)	MLUPS/K€
NEC SX8	1	66.8	11.8 GFlop/s (74%)	50	1.34
Opteron 2.4GHz/IB	1	4.3	1.1 GFlop/s (23 %)		
Opteron 2.4GHz/IB	2	8.5	2.2 GFlop/s (23 %)	6	1.42
Xeon 3.2 GHz/IB	1	4.7	1.2 GFlop/s (19 %)		
Xeon 3.2 GHz/IB	2	4.8	1.2 GFlop/s (10 %)	4.5	1.07
SGI Altix/1.6 GHz	1	8.3	2.1 GFlop/s (33 %)		
SGI Altix/1.6 GHz	2	9.7	2.4 GFlop/s (19 %)	10	0.97
IBM BG/L	*2*	*~2**	*~0.5 GFlop/s (9%)*	*??*	*??*

1 NEC SX8 CPU ~ 60 BG/L CPUs

* *Own estimation based on Read/Modify/write bandwidth of 1 BG/L CPU ~ 0.5 Opteron (cf. "A Performance and Scalability Analysis of the BlueGene/L Architecture", K. Davis et al. Proc. SC2004)*

Application Performance

Large Eddy Simulation On Curvilinear Coordinates

LESOCC - PD Dr. Breuer, Computational Fluid Dynamics, FAU
HLRS-Teraflop-Workbench

P. Lammers, G. Wellein, Th. Zeiser, G. Hager, and M. Breuer,
to be published in Proceedings of the 2nd Teraflop-Workshop at HLRS.
Have the vectors the continuing ability to parry the attack of the killer micros?

Application LESOCC from TFlops-Workbench: Method

- **LESOCC** (**L**arge **E**ddy **S**imulation **O**n **C**urvilinear **C**oordinates)
- **Navier-Stokes solver** (incompressible fluid)
- **3-D finite volume approach**
  - Curvilinear body-fitted coordinate system
  - Non-staggered (cell-centered) grid arrangement
  - Block-structured grids
- **Spatial discretization**
  - Viscous fluxes: central differences
  - Convective fluxes: five different schemes, central diff. **CDS-2**
- **Temporal discretization**
  - Predictor step (moment. eqns.): low-storage Runge-Kutta scheme, $O(\Delta t^2)$
  - Corrector step (pressure correction equation): SIP solver (ILU)
- **Pressure-velocity coupling**: Momentum interpolation of Rhie & Chow
- **High-performance computing techniques**
  - Highly vectorized
  - Parallelized by domain decomposition and explicit message passing
  - Vector-parallel computers and SMP-clusters (Hitachi, NEC-SX, SR8k-F1)

By courtesy of PD Dr. Breuer, LSTM

Application performance LESOCC

- LESOCC is being used for more than 10 years
- SIP solver vectorization in 1980's

Sustained performance of LESOCC over time (qualitative numbers)

- NEC SX-4: 1 GFlop/s (50% Peak)
- Fujitsu VPP 700 (1996-2004): 0.92 GFlop/s (41% Peak)
- Itanium2 (1.3 GHz) (2003-2008): 1 CPU of VPP 700
- Intel Xeon (2.66 GHz): ~0.5 GFlop/s
- Hitachi SR 8000 (1 node): 3 GFlop/s
- Hitachi SR 8000 (16 node): 41 GFlop/s (Parallel Eff.: ~86%)

First Measurements on NEC SX6+ / SX8 (simple test case)

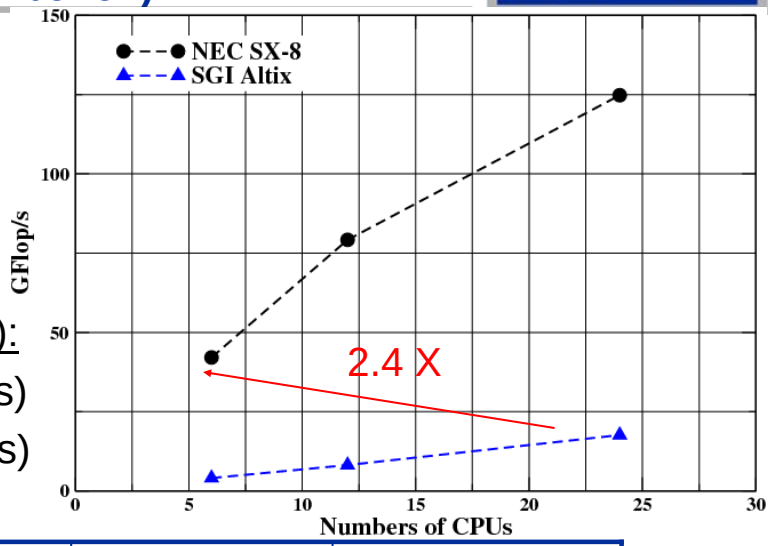
- NEC SX6+ (1 CPU): 4.7 GFlop/s
- NEC SX8 (1 CPU): 8.3 GFlop/s

Same Executable:
Speed-Up ~ 1.77

Application performance
LESOCC (HLRS-TFlops-Workbench)

Strong Scaling:

Boundary layer flow over a flat plane (11x 10^6 CVs)



SIP-solver (Max. performance):

NEC: *hyperplane* (~5-6 GFlops)

Altix: *3D* (1 GFlops per 2 CPUs)

	#CPU	SIP-solver Time [%]	SIP-solver GFlops/CPU	LESOCC GFops/CPU
SGI Altix	24	25.0	0.39	0.73
NEC SX8	6	31.5	3.25	7.02
NEC SX8	24	33.6	2.50	5.20

CFD kernel: SIP-Solver

(Part of LRZ benchmark suite)

- Solving $\mathbf{A\ x = b}$ for finite volume methods can be done by Strongly-Implicit-Procedure (SIP) according to Stone
- SIP-solver is widely used:
  - LESOCC, FASTEST, FLOWSI (Institute of Fluid Mechanics, Erlangen)
  - STHAMAS3D (Crystal Growth Laboratory, Erlangen)
  - CADiP (Theoret. Thermodynamics &Transport Processes, Bayreuth)
- SIP-Solver: 1) Incomplete LU-factorization
  2) Series of forward/backward substitutions
- Basic toy program available at: `ftp.springer.de` in `/pub/technik/peric` (M. Peric)
- Highly Optimized kernels for various architectures

CFD kernel: SIP-solver
Data-dependencies & Implementations

Basic data-dependency: **(i,j,k)←{(i-1,j,k);(i,j-1,k);(i,j,k-1)}**

```
do k = 2 , kMax
   do j = 2 , jMax
      do i = 2 , iMax
            RES(i,j,k) ={RES(i,j,k)-LB(i,j,k)*RES(i,j,k-1)
$         - LW(i,j,k)*RES(i-1,j,k)- LS(i,j,k)*RES(i,j-1,k)
$         }*LP(i,j,k)
      enddo
   enddo
enddo
```

3-fold nested loop (3D): (i,j,k)

- Data-locality (Caches !)
- Shared memory parallelization: *Pipeline parallel processing*
- Comp. intensity: 1.3 Flop/Word
  –> Max. Performance: 1 GFlop/s at 6.4 GByte/s bandwidth

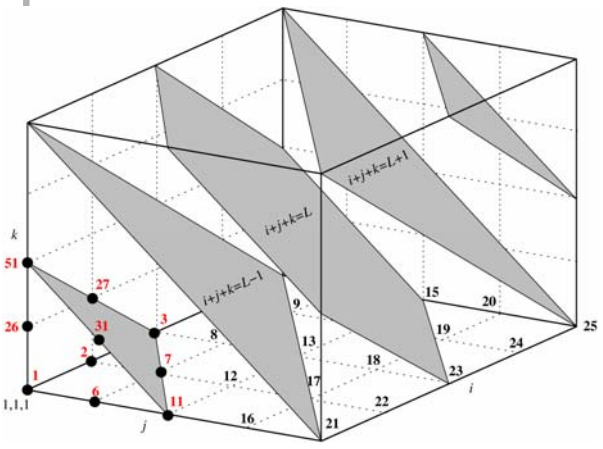
SIP-solver:
Resolving Data Dependencies With Hyperplanes

Basic data dependency:

(i,j,k) ⟵ {(i-1,j,k);(i,j-1,k);(i,j,k-1)}

Define Hyperplane: **i+j+k=const**

- non-contiguous memory access
- shared memory parallelization /vectorization of innermost loop

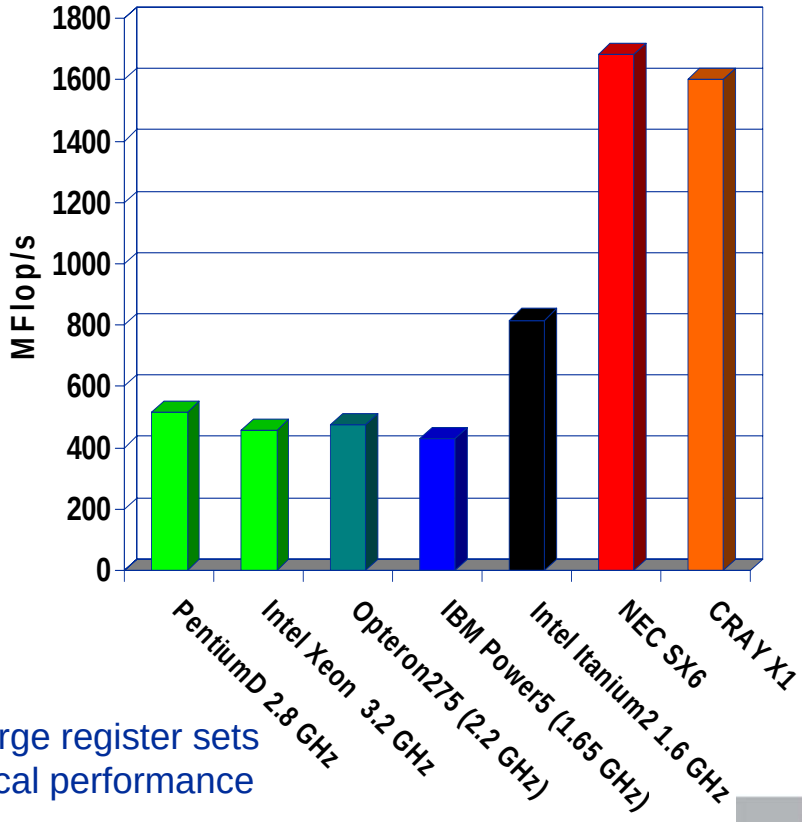



```
do l=1,hyperplanes
  n=ICL(l)
  do m=n+1,n+LM(l)
    ijk=IJKV(m)
    RES(ijk)=(RES(ijk)-
$     LB(ijk)*RES(ijk-ijMax)-
$     LW(ijk)*RES(ijk-1)-
$     LS(ijk)*RES(ijk-iMax))
$     *LP(ijk)
  enddo
enddo
```

CFD kernel: SIP-solver
Data-dependencies & Implementations

Benchmark:

- Lattice: 91^3
- 100 MB
- 1 ILU
- 500 iterations
- Opteron: ifort
- CRAY X1: 1 MSP
- IBM: A story of its own…
  (IBM: > 1 GFlop/s on p575)

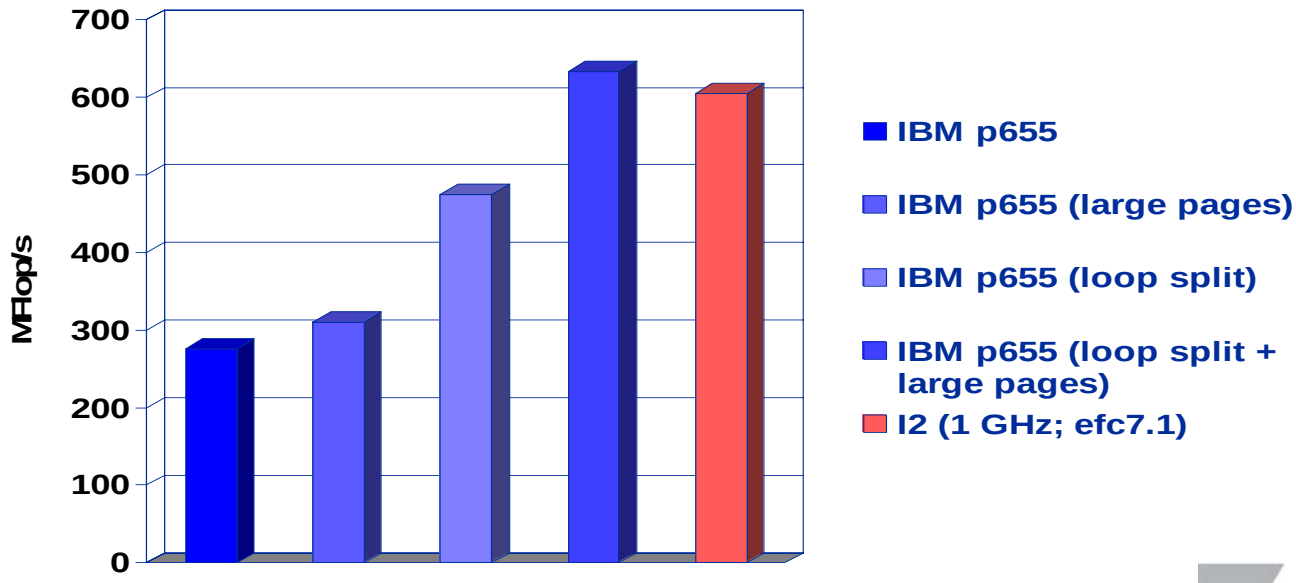


Intel Itanium2 benefits from large register sets
& is very close at the theoretical performance

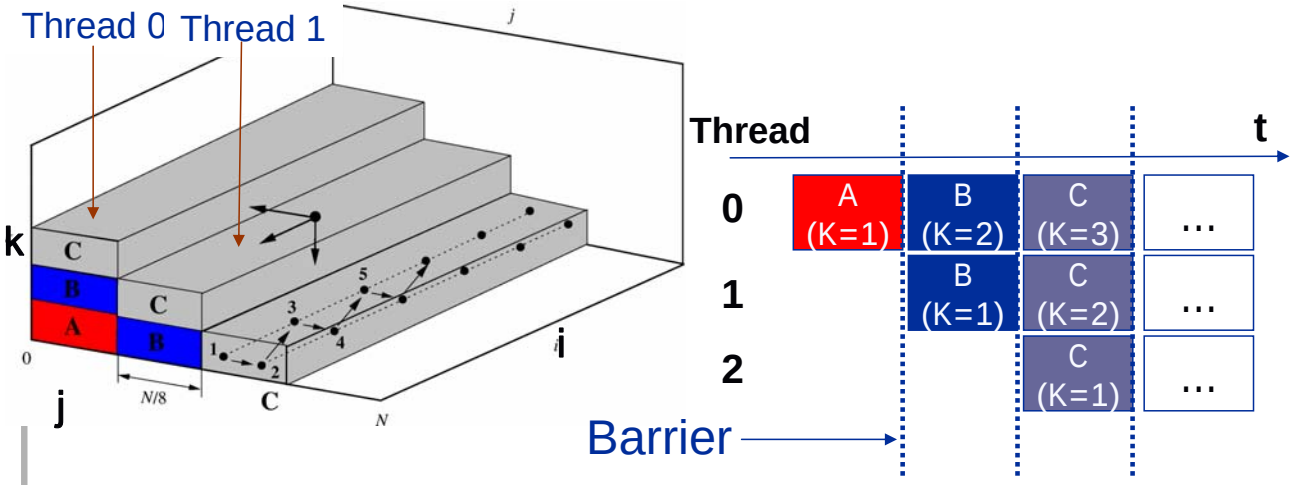
SIP-solver:
Implementations & Single Processor Performance

size=89^3 (100 MB)
IBM improvements on Power4:
split single loop in 4 seperate loops; use large pages

It`s hard to reproduce these numbers on production systems



CFD kernel: SIP-solver
Pipeline Parallel Processing



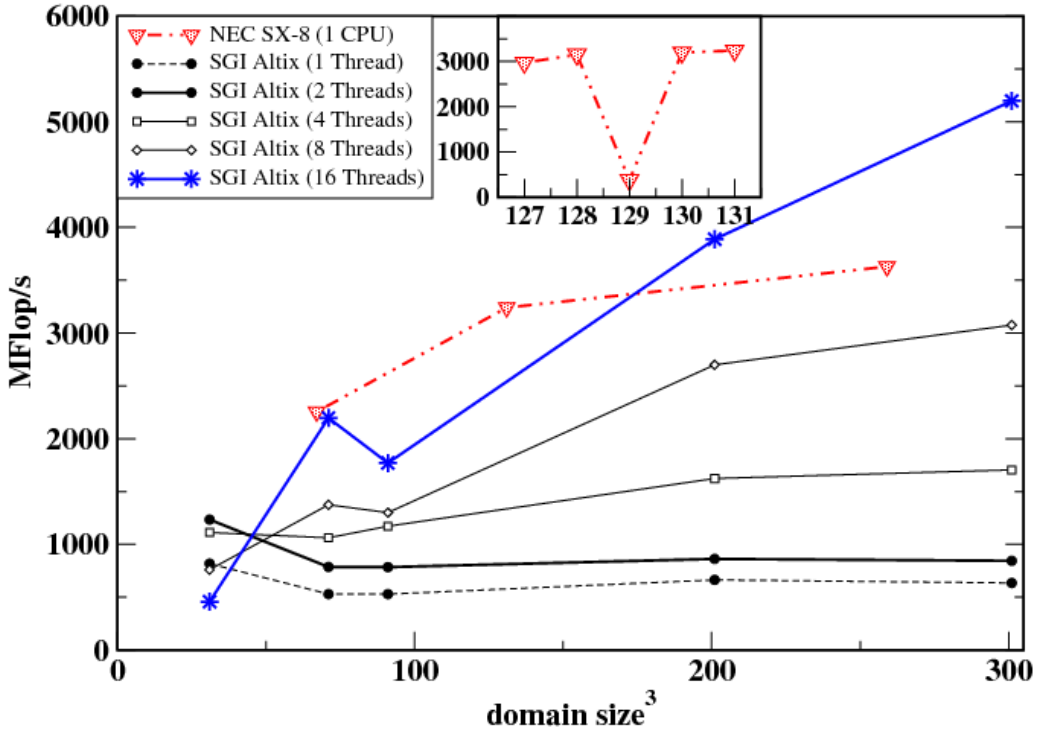
- Split up j-loop in chunks of equal size for each thread
- Split up k-direction in blocks of equal chunks
- Pipelining in k-direction: Only one k-Index is active at a fixed time
- Efficient parallelisation if `kMax, jMax >> #Threads`

CFD kernel: SIP-solver
Pipeline Parallel Processing using OpenMP

```
$omp parallel private(….)
do l =2, kMax+numThreads-2,1
   threadID=OMP_GET_THREAD_NUM()
   k = l – threadID
   if((k.ge.2).and.(k.le.kMaxM)) then
      do j = jS(threadID),jE(threadID)
         do i = 2 , iMax
            RES(i,j,k) = {RES(i,j,k)-
$             LB(i,j,k)*RES(i,j,k-1) …   ! Same thread
$       …     LS(i,j,k)*RES(i,j-1,k) …}  ! threadID-1 in
         enddo                           ! in prev. l-iter
      enddo
   endif
$omp barrier
enddo
$omp end parallel
```

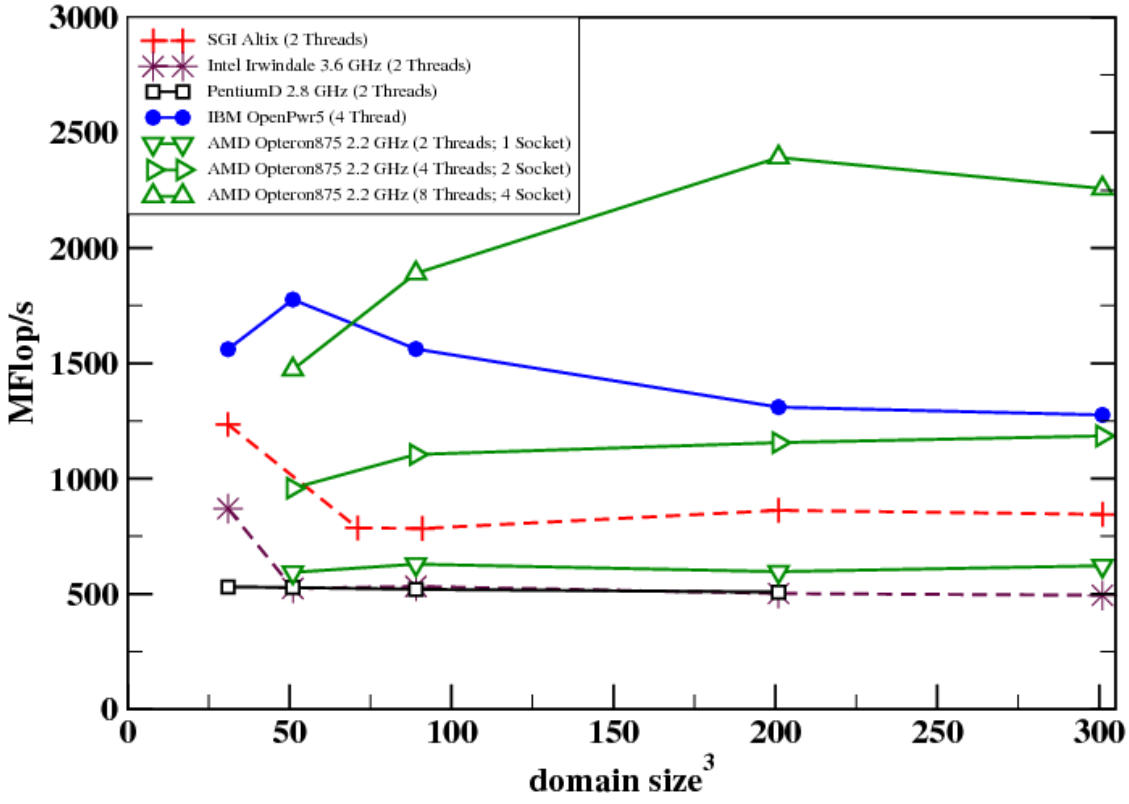
CFD kernel: SIP-solver
Pipeline Parallel Processing using OpenMP

SGI Altix vs. NEC SX8



CFD kernel: SIP-solver
Pipeline Parallel Processing using OpenMP

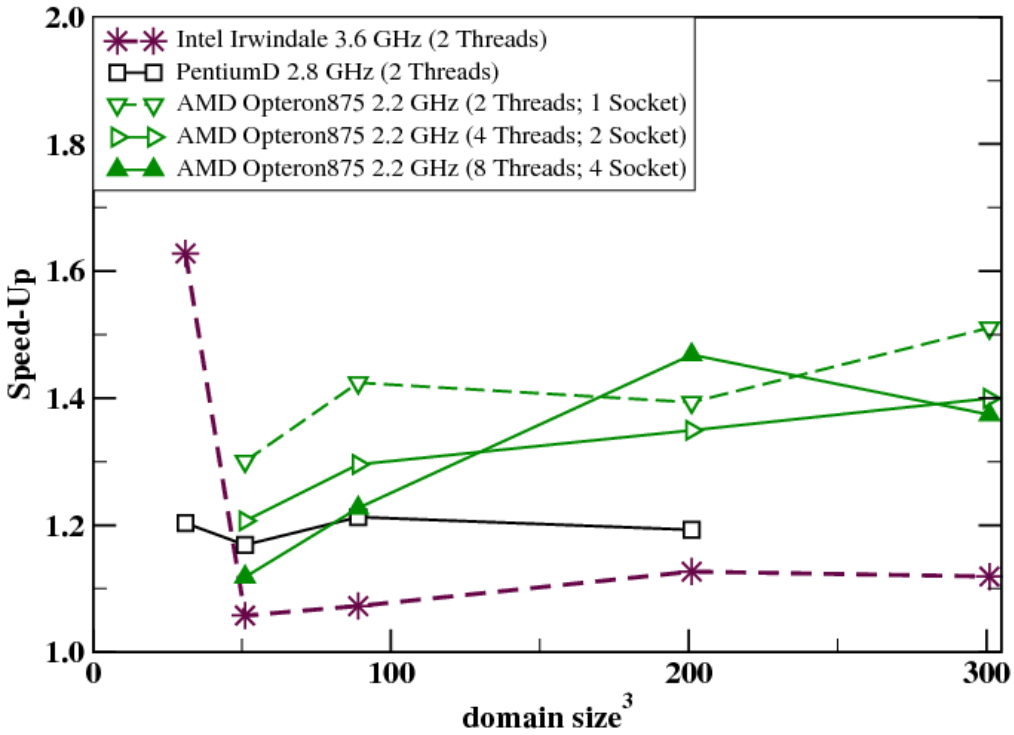
Dual-Cores vs. classical Intel architictures



CFD kernel: SIP-solver
Pipeline Parallel Processing using OpenMP

Benefit of second core on AMD Opteron & PentiumD - Speed-Up:

P(2,N) / P(1,N) [P(I,N): perf. using N sockets & running I cores/socket]



2nd core helps Opteron to make better use of memory interface

Speed-Up: ~20-30% compared to a pure seq. code

Application Performance

Density Matrix Renormalization Group computations for Highly Correlated Quantum Systems

Prof. H. Fehske, Theoretical Physics, Univ. Greifswald
Dipl. Phys. G. Hager, Dr. G. Wellein, RRZE, Erlangen

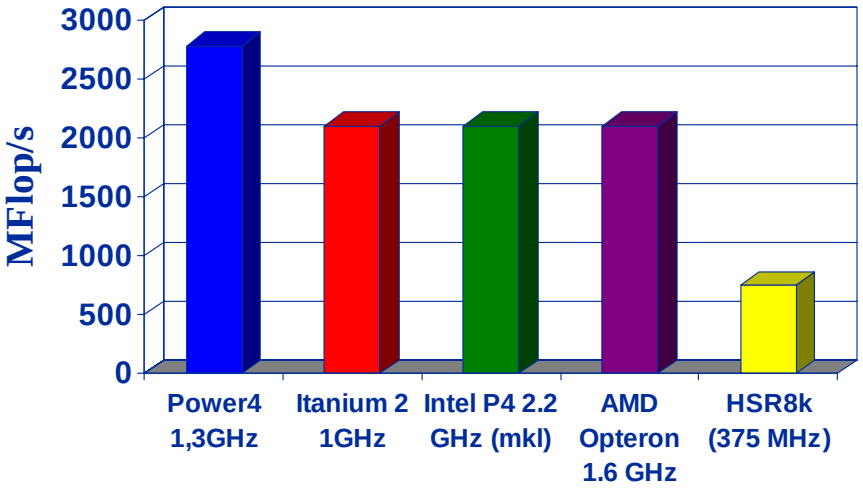
(Part of LRZ benchmark suite)

G. Hager, E. Jeckelmann, H. Fehske, and G. Wellein, J. Comp. Phys., 194-2, 795-808 (2004).
Parallelization strategies for density-matrix renormalization group algorithms on shared-memory systems

Quantum Physics Application: DMRG (C++ & OpenMP)

Density Matrix Renormalization Group:

- Calculate ground state and excitations of strongly correlated quantum systems (physics/chemistry)
  - Large C++ program with DGEMM as core routine; OpenMP parallel
  - Severe problems using complex C++ constructs & OpenMP on



Requirements:

•Large shared memory

•Peak Performance

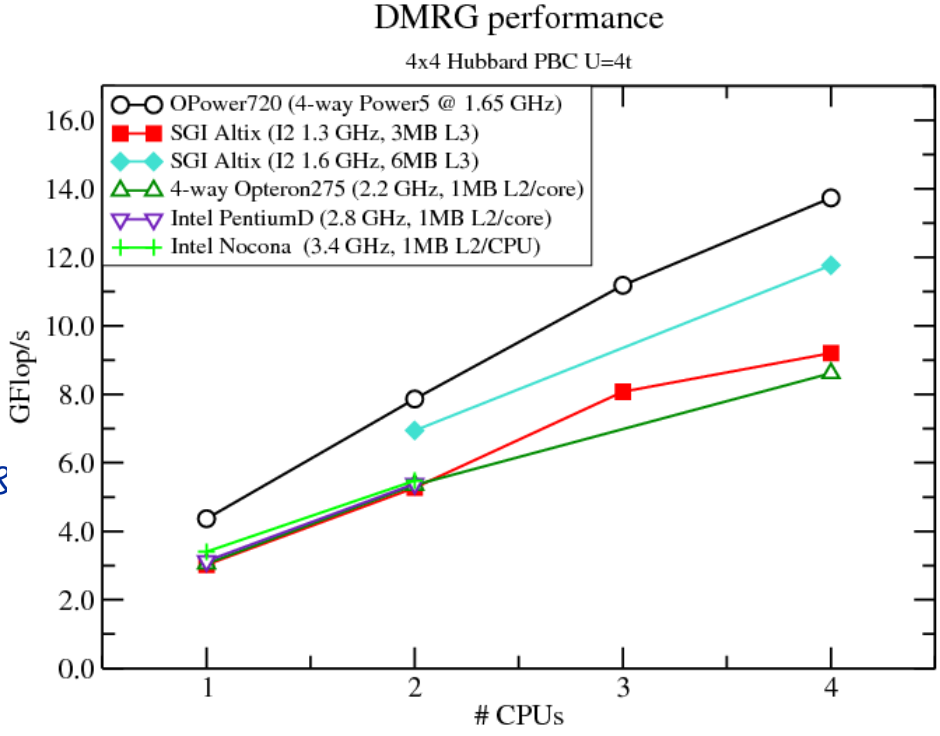
•1-16 Threads

Target architecture:

• IBM p690 or SGI Altix

Quantum Physics Application: DMRG (C++ & OpenMP)

- OpenMP parallelization of mature C++ code (S. White, Irvine)
- 4-16 threads
- May run for weeks & require > 100 GByte
- Benefits from Dual-Core technology



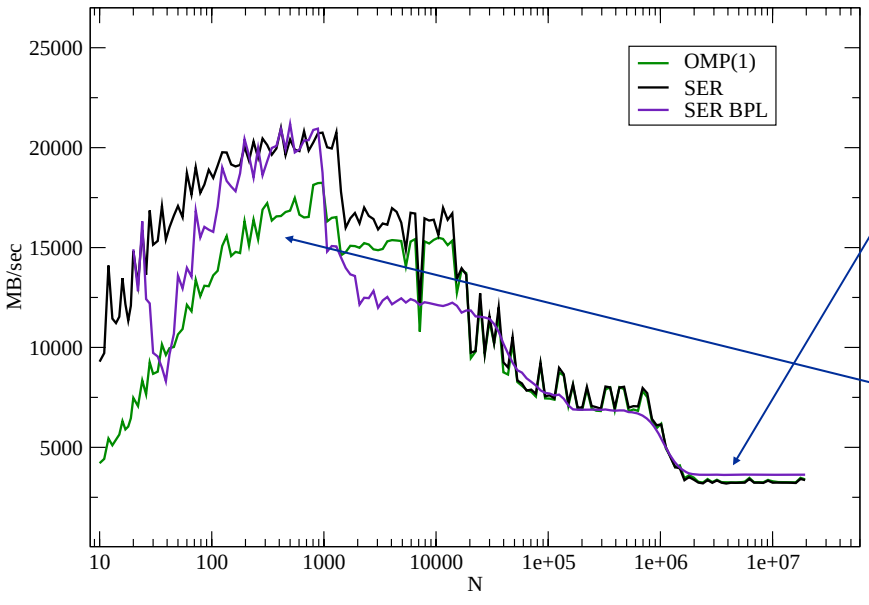
OpenPower 720

Excerpts from RRZE benchmark report

G. Hager, T. Zeiser, G. Wellein
RRZE

OpenPower 720 triads - serial

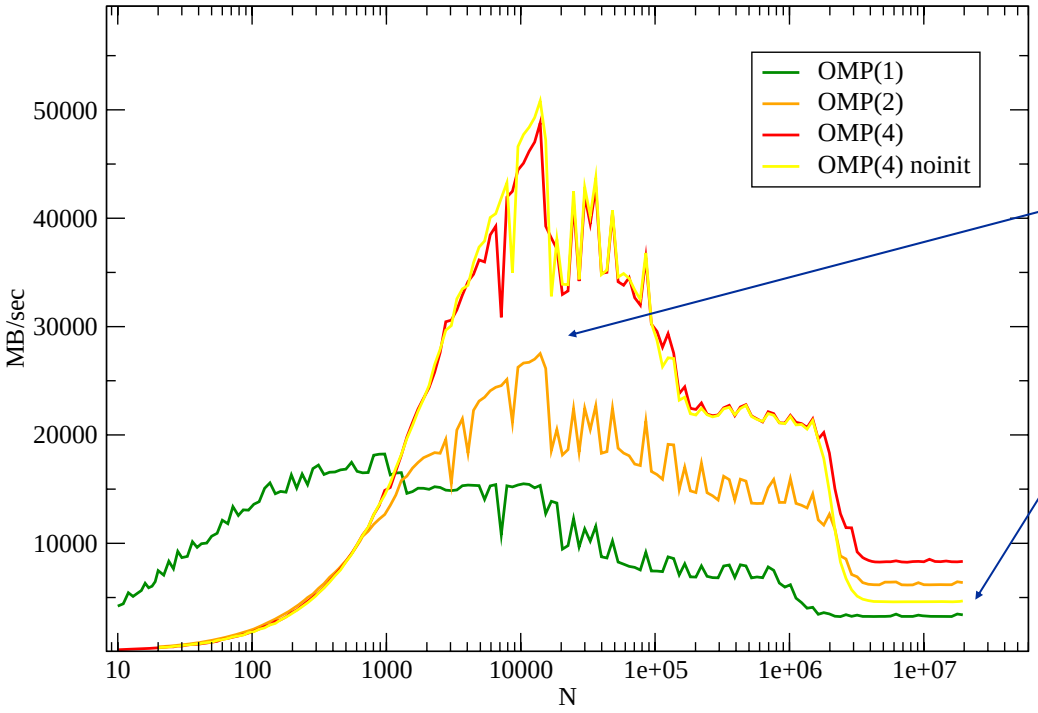
- Vector triads on OpenPower720 (2x DC Pwr5 @ 1.65 GHz)



Approx. 10% gain from block preload in memory

some in-cache performance hit due to OMP

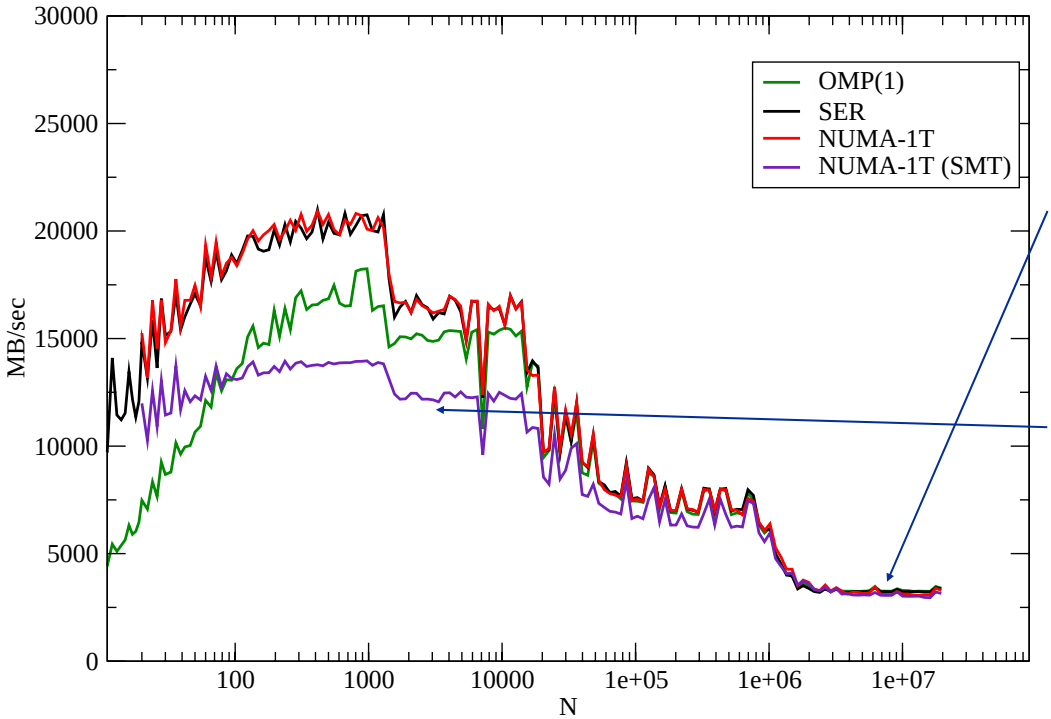
OpenPower 720 triads - OpenMP



perf drop @ L2 bank size independent of thread #

30-40% BW gain by using more than 1 core per memory channel

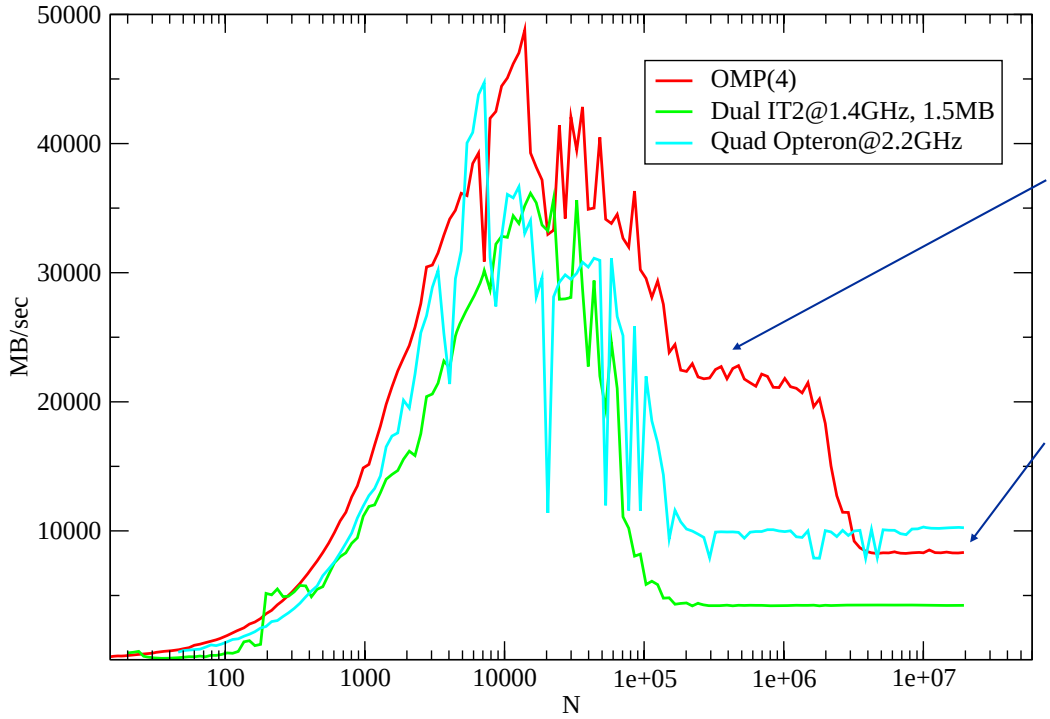
OpenPower 720 triads – NUMA effects



hardly any BW penalty for NUMA access

SMT very bad for 1-T in cache performance

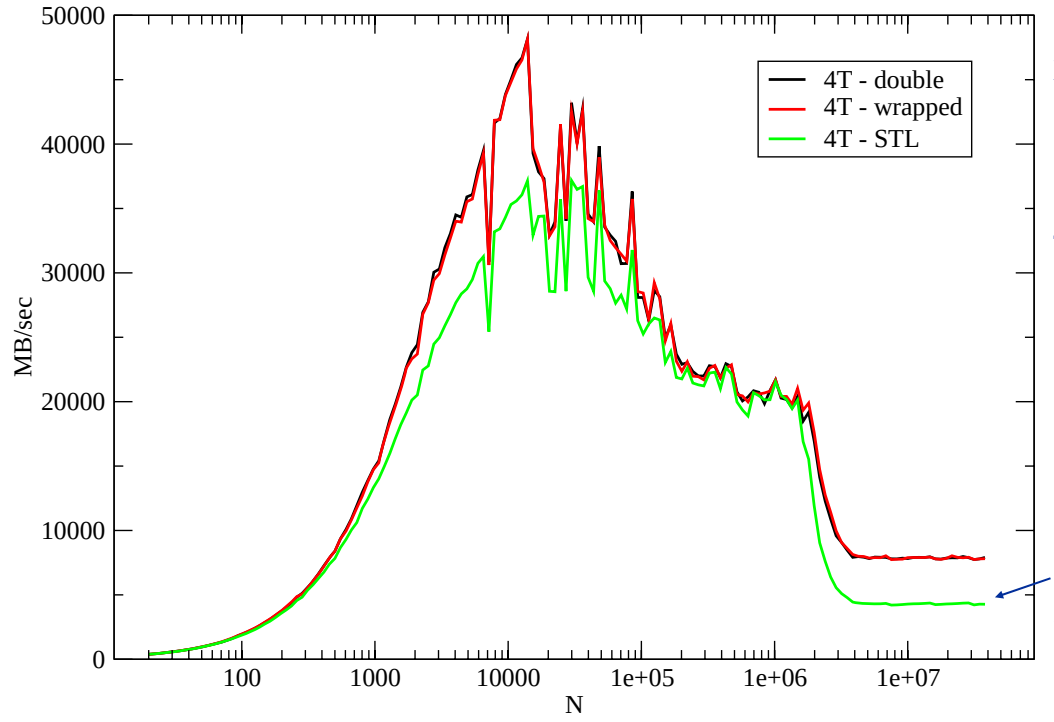
OpenPower 720 triads – architectures



OP720: very good in-cache BW, large L3

memory BW not up to expectations, still outperformed by quad Opteron

OpenPower 720 triads – xlC rulez!



xlC is only compiler that can optimize "wrapped" triad to full performance!

(apart from MIPSPro…)

here: no NUMA-aware STL allocator

Aktuelle Experimente mit Compilern

G. Hager
RRZE

Compilerversionen im Einsatz am RRZE

- **Intel 9.0-02X + 8.1-02X auf IA32/EM64T/x86_64/IA64**
  - **stabile Situation, wenig Bugs**
  - **Probleme mit Optimierungen unter OpenMP (nur bei C++?)**
- **PGI 6.0-5**
  - **Unter OpenMP besser als IA64-Compiler aber schlechter als EM64T-Compiler**
- **Pathscale 2.2.1**
  - **Testlizenz mittlerweile abgelaufen, Beschaffung läuft**
  - **gut bei Standard-Code, schlägt sich gut bei OpenMP**
  - **etwas buggy, Abhängigkeit von GCC**
- **MIPSPro 7.41 (IRIX/MIPS)**
  - **einfach nur cool.**
- **Stepanov-Test?**

C++-Triade mit STL und OpenMP

```
typedef vector<double,NUMA_Allocator<double> > vdn;
double stl_triad(vdn &a, const vdn &b,...) {
```

V1

```
#pragma omp parallel for schedule(static)
for(i=0; i<len; ++i)
   a[i] = b[i] + c[i] * d[i];
...
```

V2

```
vector<double>::iterator ai = a.begin();
vector<double>::const_iterator bi = b.begin();
...
#pragma omp parallel for schedule(static)
for(i=0; i<len; ++i)
   ai[i] = bi[i] + ci[i] * di[i];
```


C++-Triade: Penalties (out of cache)

	STL + operator[]	STL + iterator	STL + operator[] + OMP (V1)	STL + iterator + OMP (V2)	double[] + OMP
Intel V9 IA64*	0.50	0.99	0.25	0.28	0.98
Intel V9 EM64T	0.78	0.80	0.64	0.79	1.00
PGI x86_64	0.53	0.90	0.47	0.68	0.79
Pathscale x86_64§	0.87	0.87	0.81	0.87	1.00
MIPSPro MIPS	0.78	1.00	0.84	0.95	1.00

* `#pragma ivdep` vor Schleife verhindert OMP-Parallelisierung

§

Probleme

- **Intel V9-02X IA64:**

```
#pragma ivdep
#pragma omp parallel for schedule(static)
   for(....)
```

OMP #pragma wird ignoriert!

- **Pathscale 2.2.1:**

Abhängigkeit von GCC hat Fehler zur Folge, für die Pathscale nichts kann

Probleme

- **Pathscale 2.2.1 (Forts.): In `class NUMA_Allocator<T>`:**

```
   pointer allocate(size_type numObjects, const void *lh=0) {
      size_type ofs,len = numObjects * sizeof(value_type);
25    void *m = malloc(len); char *p = static_cast<char*>(m);
26    int i,pages = len >> PAGE_BITS;
27 #pragma omp parallel for schedule(static) private(ofs)
28    for(i=0; i<pages; ++i) {
29      ofs = static_cast<size_t>(i) << PAGE_BITS;
30      p[ofs]=0;
31    }
      return static_cast<pointer>(m);
    }
```

GCC 3.3:

```
nalloc.h: In member function `T*
          NUMA_Allocator<T>::allocate(long unsigned int,
             const void*)':
          nalloc.h:26: error: parse error before `;' token
          nalloc.h:27: error: parse error before `;' token
```

GCC 3.4: kein Problem!

Experience with the latest VTune releases

T. Zeiser, G. Hager
RRZE

Intel VTune

- Released versions
  - Intel VTune 3.0 Linux
    - Command line version (CLI)
    - Command line version with GUI
    - Eclipse GUI (only 32-bit)
  - Intel VTune 7.2 Windows
    - Native for MS Windows
    - Remote data collection (RDC) => Linux
- New beta version
  - Intel VTune 8.0 beta Linux (beta program: 08/2005…12/2005)
    - Command line version (CLI)
    - Command line version with GUI
    - Eclipse GUI (only 32-bit)
    - Remote data collection (RDC with access control)
- VTune = Linux kernel module + database services + user tools

Intel VTune 3.0 / 7.2

- Kernel modules and modules compatible between Windows and Linux versions
- Updated versions of the kernel modules unofficially available
- Only very limited support for kernel 2.6
- No access control for RDC; port 50000 open for everybody
- Typical problems:
  - Modules (programs) not found => Other32/Other64
  - Source code information not found
  - Severe dependence on kernel, glibc, compiler, … versions

New features of Intel VTune 8.0 beta Linux

- Current Windows version NOT compatible with Linux beta
- Better support for kernel 2.6
- Access control now supported for RDC (via PAM)
- Typical problems:
  - Still sometimes modules (programs) not found => Other32/Other64
  - Still some dependence on compiler versions
  - `locale` must have appropriate value, i.e. “C” instead of iso*/*UFT*
  - => otherwise sometimes segfault or similar non-specific errors when running user commands
  - CLI sensitive to certain stacksize limits
  - Parallel installation of Linux beta and old Windows-compatible RDC possible but requires much manual activity during installation
- General remarks
  - Eclipse GUI still not usable (slow, unstable, …)
  - Collected data must be stored on a local directory not NFS