OpenMP in the Works

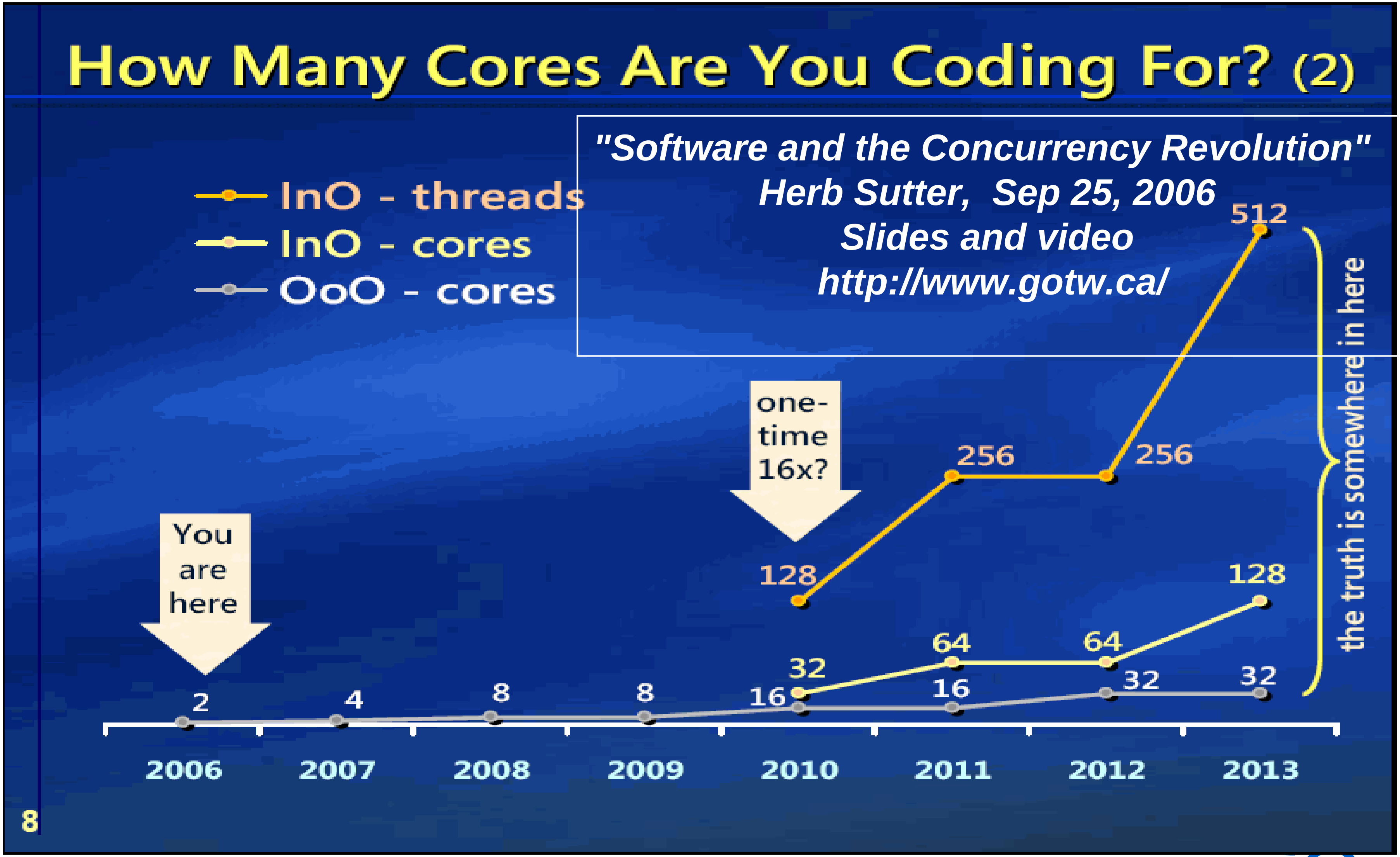
July 4, 2007, Jülich

Dieter an Mey,
Christian Terboven, Samuel Sarholz, Alex Spiegel

Center for Computing and Communication
RWTH Aachen University, Germany

`www.rz.rwth-aachen.de`
`anmey@rz.rwth-aachen.de`

Motivation



"Software and the Concurrency Revolution"
Herb Sutter, Sep 25, 2006
Slides and video
http://www.gotw.ca/

RWTH Center for Computing and Communication

Motivation

Burton Smith in "Reinventing Computing" during the ISC last week:

- use multithreaded cores to tolerate memory latency
- when latency increases, increase the number of threads

…

- We need to support multiple programming styles
  - both functional and transactional
  - data and task parallel
  - message passing and shared memeory
  - declarative and imperative
  - implicit and explicit

…

- Consequences for HPC
  - Routine combining of shared memory and message passing
- HPC will need to be reinvented along with everything else

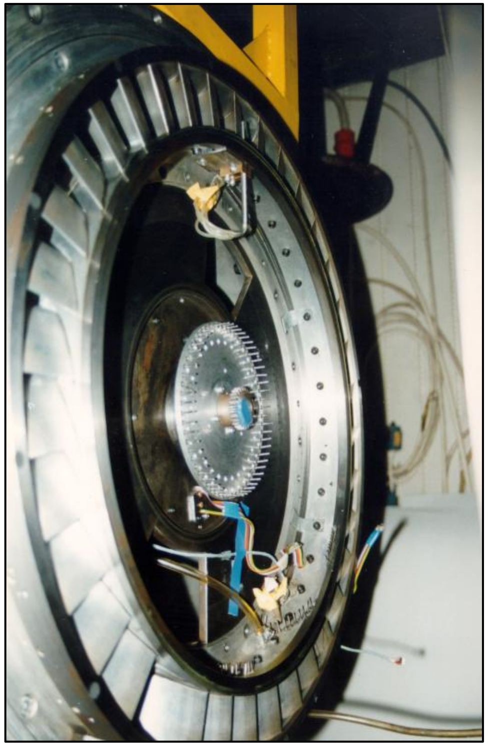
OpenMP in the Works - Overview

- Loop-Level Parallelization in Fortran
  - Autoscoping
  - Combining Autoparallelization, OpenMP, Sun Performance Library
  - Pushing Loop-Level Parallelization to the Limit
- C++ and OpenMP
  - DROPS
  - Realtime FEM for VR
- Nested Parallelization
  - Pattern Recognition
  - Critical Points
  - TFS parallelized with Parawise by PSP
  - Dynamic Thread Balancing for MPI+OMP
- OpenMP Tools / OpenMP on Windows
- CMP / CMT

Overview

- **Loop-Level Parallelization in Fortran**
  - **Autoscoping**
  - **Combining Autoparallelization, OpenMP, Sun Performance Library**
  - **Pushing Loop-Level Parallelization to the Limit**
- C++ and OpenMP
  - DROPS
  - Realtime FEM for VR
- Nested Parallelization
  - Pattern Recognition
  - Critical Points
  - TFS parallelized with Parawise by PSP
  - Dynamic Thread Balancing for MPI+OMP
- OpenMP Tools / OpenMP on Windows
- CMP / CMT

Loop Level Parallelization
PANTA 3D Navier-Stokes Solver



- 121 routines with 18497 Fortran lines are considered for **auto-parallelization**
- out of these 5 routines with 2799 Fortran lines have been parallelized **manually** with **132 OpenMP** directives (incl. cont.) and
- another 11 routines have been auto-parallelized with **Visual KAP** introducing 608 OpenMP directives.

- **1389 variables** had to be **scoped** manually
- Could be reduced to **13 with Autoscoping !**

- **Speedup: 3 with 4 threads**

Hans Thermann, Bernd Wickerath, Daniel Grates, Stephan Schmidt, Volmar,T.,Brouillet,B.,Gallus,H.E.,Benetschik,H., Institute for Turbomachinery, RWTH Aachen University

PANTA – Autoscoping a parallel loop (part 1)

```
!$omp parallel private(local_QSIK,local_QSIE)
!$omp do private (lijk,xiaxc,xiayc,xiazc,xiaxeb,xiayeb,xiazeb, &
!$omp & xibxnb,xibynb,xibznb,xibxsb,xibysb,xibzsb,xicxub,xicyub, &
      ... [...]  100 lines omitted
!$omp & dwydq1,dwydq4,dwzdq1,dwzdq4,amt,omsqr,dmtdq1,dmtdq2,dmtdq3) &
!$omp lastprivate(INE,IEE,ISE,ISW,IWW,INW,IUE,IUW,IDE,IDW,INU,IND)
      do i = is,ie
            [...]   1949 lines omitted
      end do
!$omp end do
!$omp end parallel
```

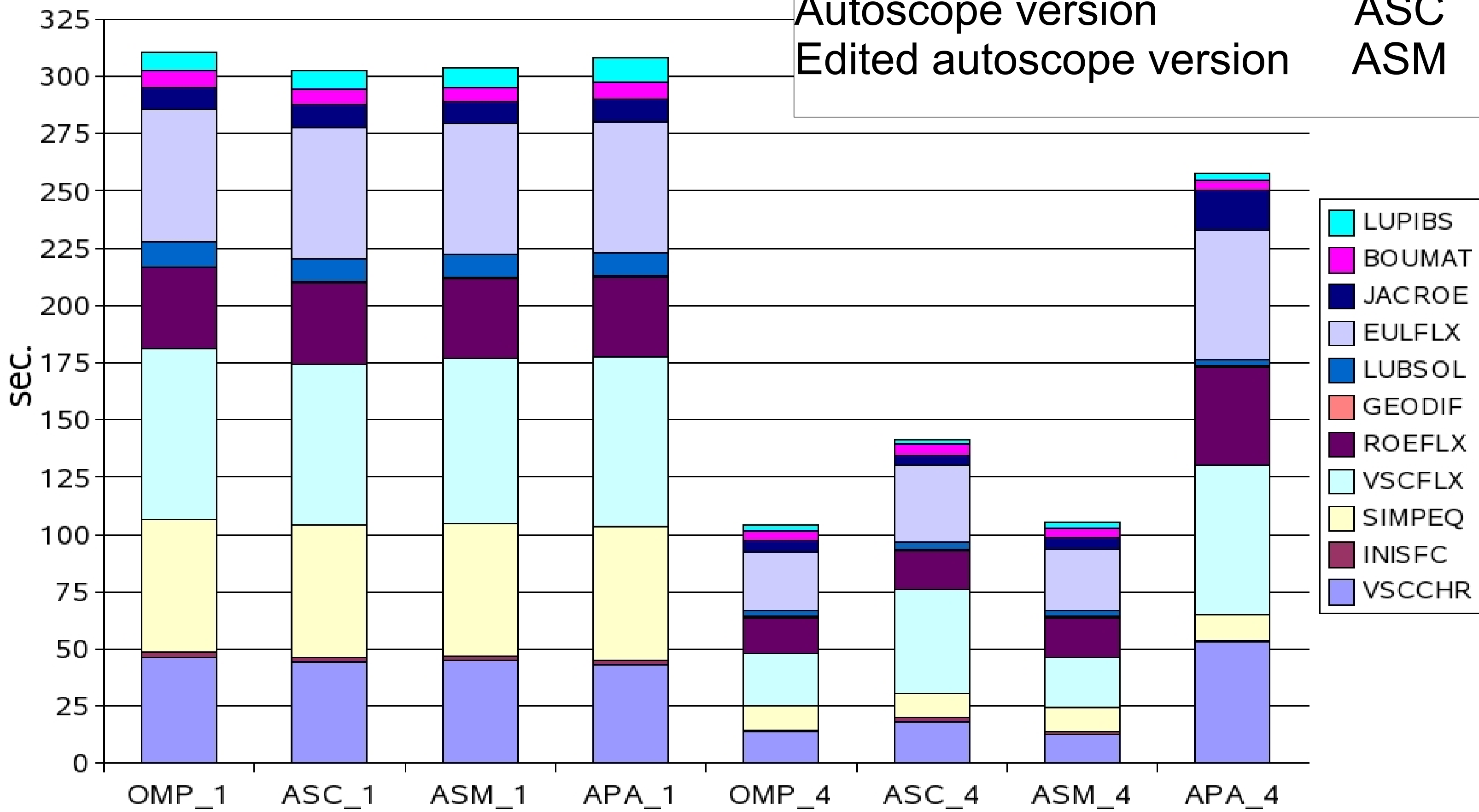
This is what a programmer typically has to add in order to parallelize a loop in a CFD code with OpenMP

```
!$omp parallel DEFAULT(__AUTO)
!$omp do
      do i = is,ie
            [...]   1949 lines omitted
      end do
!$omp end do
!$omp end parallel
```

This is what a programmer typically has to add when using Sun's **autoscoping** feature.

PANTA– Autoscoping performance comparison

(original) OpenMP version	OMP
Autoparallel version	APA
Autoscope version	ASC
Edited autoscope version	ASM



1 Thread 4 Threads

RWTH Center for Computing and Communication

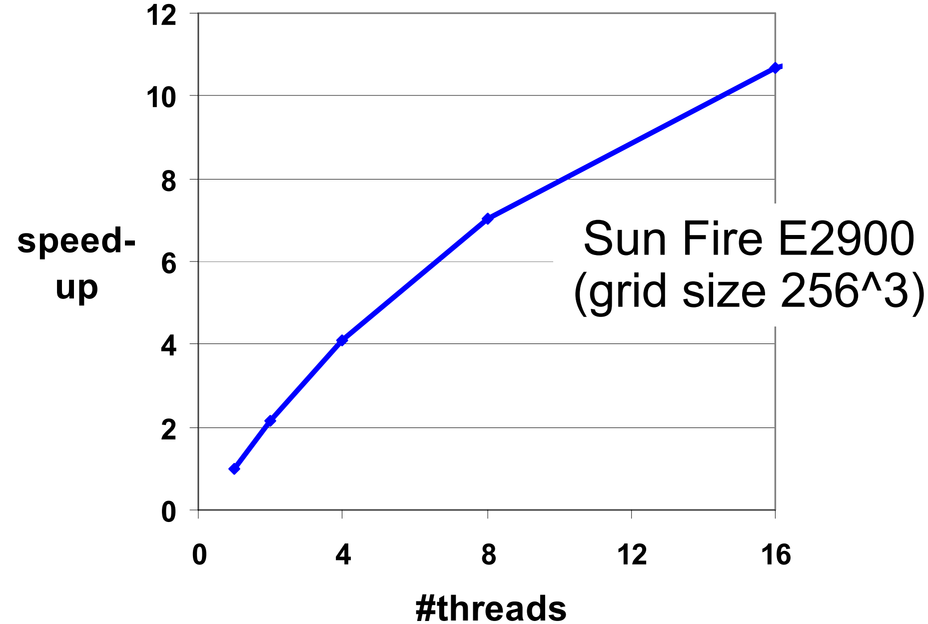
Direct Numerical Simulation (DNS) of Turbulences

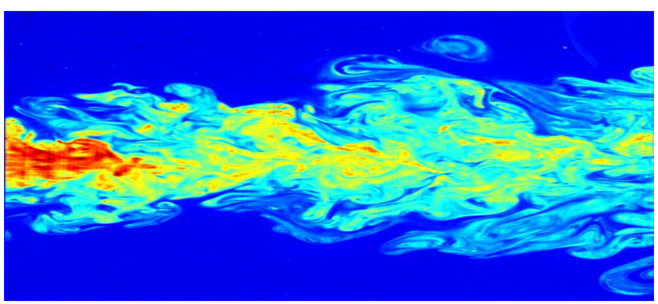
Lipo Wang, Institut für Technische Verbrennung, RWTH Aachen University

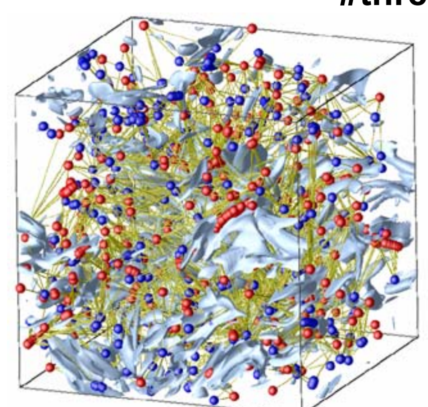
A combination of

- Sun Performance Library
  a special version of the 3D FFT routines has been parallelized upon request
- Autoparallelization
  Sun Fortran Compiler
- OpenMP directives
  Sun Fortran Compiler

leads to good scalability.





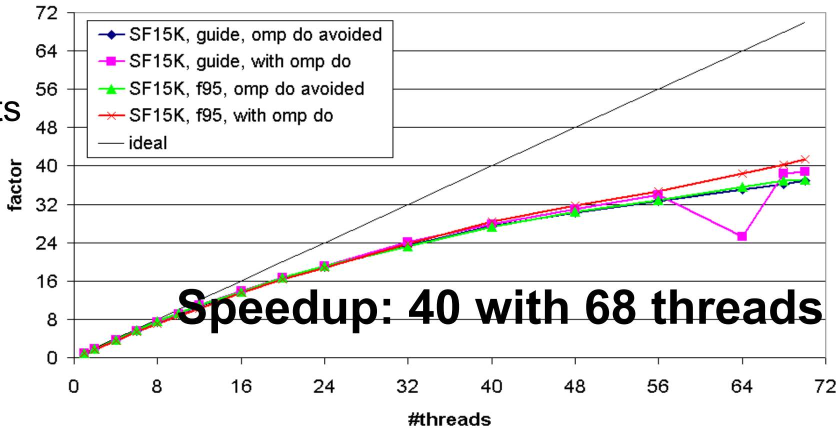


The distribution of extreme points attached with vorticity sheets

Heat Flow Simulation with FEM - ThermoFlow60

Thomas Haarmann, Wolfgang Koschel, Jet Propulsion Laboratory, RWTH Aachen University

- simulation of the heat flow in a rocket combustion chamber
- Finite Element Method
  - 200,000 cells
  - 230 MB memory footprint
  - 29000 lines of Fortran
- OpenMP Parallelelization
  - ~ 200 OpenMP directives
  - 69 parallel loops
  - 1 main parallel region **(orphaning)**
    - Explicite worksharing by precalculating loop limits to improve load balancing
- ***Time for production run cut down from 2 weeks to 1 day on 16 CPUs and to 9 hours on a 72 CPUs***



Overview

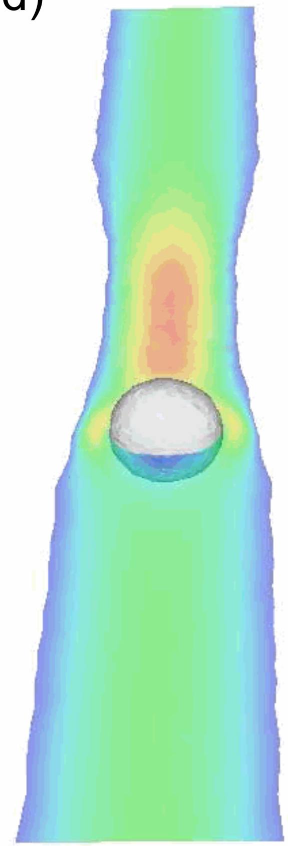
- Loop-Level Parallelization in Fortran
  - Autoscoping
  - Combining Autoparallelization, OpenMP, Sun Performance Library
  - Pushing Loop-Level Parallelization to the Limit
- **C++ and OpenMP**
  - **DROPS**
  - **Realtime FEM for VR**
- Nested Parallelization
  - Pattern Recognition
  - Critical Points
  - TFS parallelized with Parawise by PSP
  - Dynamic Thread Balancing for MPI+OMP
- OpenMP Tools / OpenMP on Windows
- CMP / CMT

DROPS: Introduction

- Numerical simulation of two-phase flow
- The two-phase flow is modeled by the instationary and non-linear Navier-Stokes equation
- So-called level set function is used to describe the interface between the two phases
- DROPS is written in C++
  (object-oriented, templates, STL, compile-time polymorphism, nested solvers with its preconditioners, smoothers etc.)

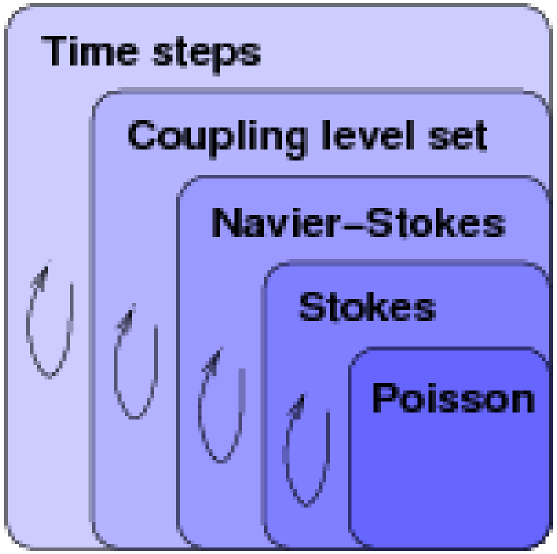
- (Adaptive) Tetrahedral Grid Hierarchy
- Finite Element Method (FEM)

Example:
Silicon oil drop in D_2O
(fluid/fluid)



DROPS: Nested Solvers

- Time integration by fractional step method
- Fixed point iteration for the decoupled Navier-Stokes and the advection equations for the level set function
- Fixed point iteration for non-linear convection term in the Navier-Stokes equations
- Stokes solvers: Uzawa, Schur, MinRes, **GMRES**
- Inner solvers for Poisson-type problems.
  - preconditioned conjugate gradient (**PCG**)
  - multi-grid (MG)
- Preconditioners / smoothers: **Jacobi** or **SSOR**



The GMRES and PCG solvers were employed and parallelized in this work

RWTH
Center for Computing and Communication

DROPS: Parallelization

```
PCG(const Mat& A, Vec& x, const Vec& b,
    const PreCon& M, int& max_iter,
    double& tol)
{
  Vec p(n), z(n), q(n), r(n);
  […]
  for (int i=1; i<=max_iter; ++i) {
    […]
    q = A * p;
    double alpha = rho / (p*q);
    x += alpha * p;
    r -= alpha * q;
    […]
```

```
y_Ax_par(&q.raw()[0],
   A.num_rows(), A.raw_val(),
   A.raw_row(), A.raw_col(),
   Addr( p.raw()));
```

```
#pragma omp for reduction
                (+:alpha_sum)
for (long j=0; j<n; j++)
   alpha_sum += p[j]*q[j];

#pragma omp single {
   alpha = rho/alpha_sum;
}
```

```
#pragma omp for
for (long j=0; j<n; j++){
   x[j] += alpha * p[j];
   r[j] -= alpha * q[j];
}
```

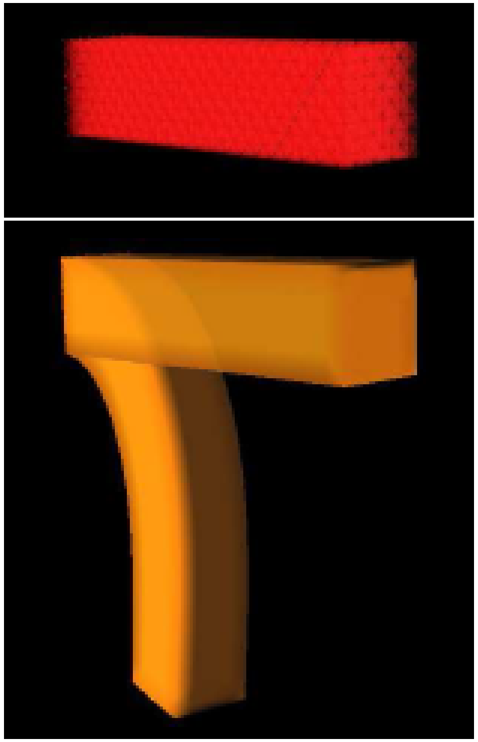

RWTH Center for Computing and Communication

OpenMP and C++

- A clean C++ object oriented coding style may be very helpful for OpenMP parallelization:
  - Encapsulation prohibits unintended data dependencies and thus simplifies data dependency analysis
  - Class members are typically local and therefore private by default which leads to a good locality in combination with first touch policy
  - (static variables would have to be protected by critical regions)
- BUT: there are several issues with OpenMP and C++
  - Compilers (may) have problems with OpenMP C++ codes
  - Non-POD types not well support (privatization, reduction)
  - Parallelization of STL Iterator loops not directly possible
  - Several issues with writing parallel libraries with OpenMP
- Still the combination works in practice, as we can see now …

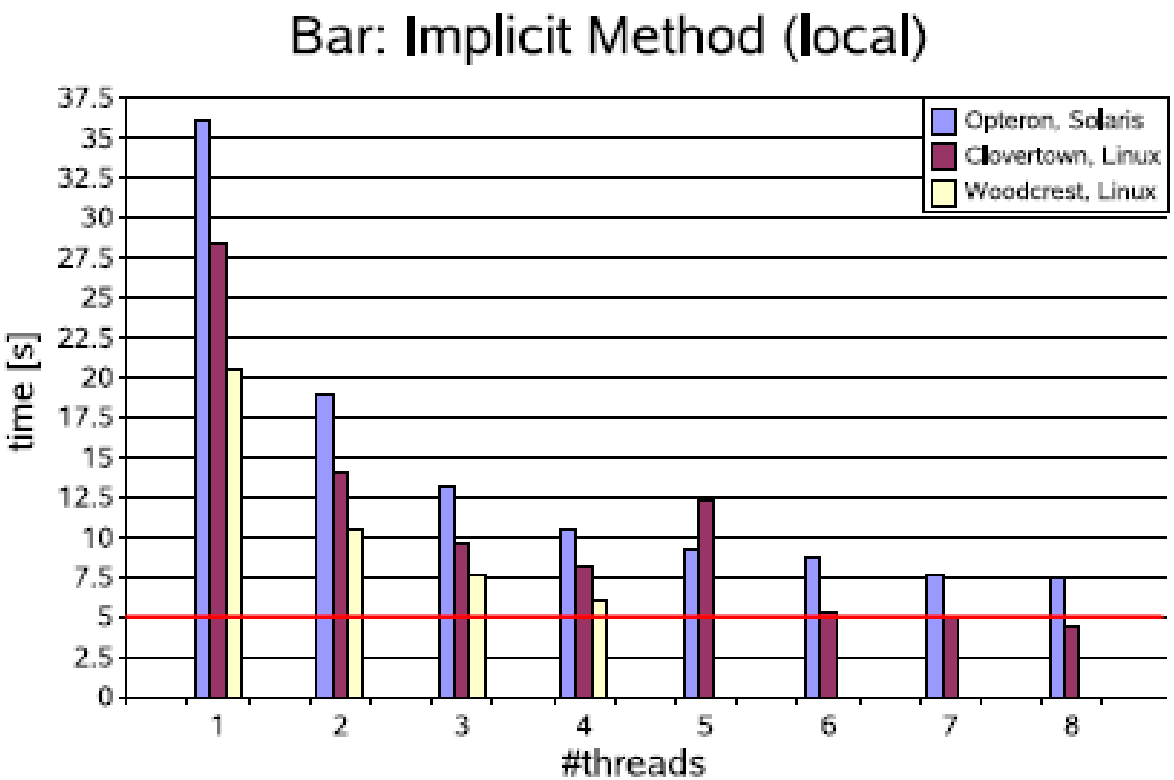
VRFEM: Realtime FEM

- VRFEM: Lenka Jerabkova, Center for Computing and Communication
- Physically based simulation is an indispensable component of many interactive virtual environments. Main challenge: realtime requirement.
- Higher computational costs than methods typically used e.g. in computer games.
- Realtime cannot be achieved using sequential approaches … performance improvement of single thread slows to increase



VRFEM: Speedup on Multicore

- The presented algorithm has been parallelized with focus on recent multicore architectures:
  - Red bar: realtime requirement, only reached on two-socket quadcore (Clovertown) system (still Pizza box)



Up to 12500 FE mesh elements, noticeable improv. to approx. 3000 elements with serial algorithm

RWTH Center for Computing and Communication

Overview

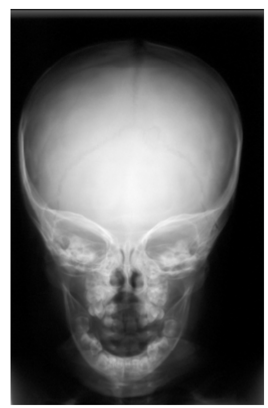
- Loop-Level Parallelization in Fortran
  - Autoscoping
  - Combining Autoparallelization, OpenMP, Sun Performance Library
  - Pushing Loop-Level Parallelization to the Limit
- C++ and OpenMP
  - DROPS
  - Realtime FEM for VR
- **Nested Parallelization**
  - **Pattern Recognition**
  - **Critical Points**
  - **TFS parallelized with Parawise by PSP**
  - **Dynamic Thread Balancing for MPI+OMP**
- OpenMP Tools / OpenMP on Windows
- CMP / CMT

FIRE: Nested Parallelization with OpenMP

- FIRE = Flexible Image Retrival Engine (written in C++)

  Thomas Deselaers, Daniel Keysers, RWTH I6 (computer science 6): Human Language Technology and Pattern Recognition

- compare the performance of common features on different databases
- analysis of correlation of different features



FIRE: Nested Parallelization with OpenMP

$$D(Q,X) := \sum_{m=1}^{M} w_m \cdot d_m(Q_m, X_m)$$

- Q: query image, X: set of database images
- Q_m, X_m: m-th feature of Q and X
- d_m: distance measure, w_m: weighting coefficient
- Return the k images with lowest distance to query image

- Well-suited for shared memory parallelization because of large image database

- 3 Levels to employ parallelization:
  - Process multiple query images in parallel
  - Process database comparison for one query image in parallel
  - Computation of distances might be parallelized as well

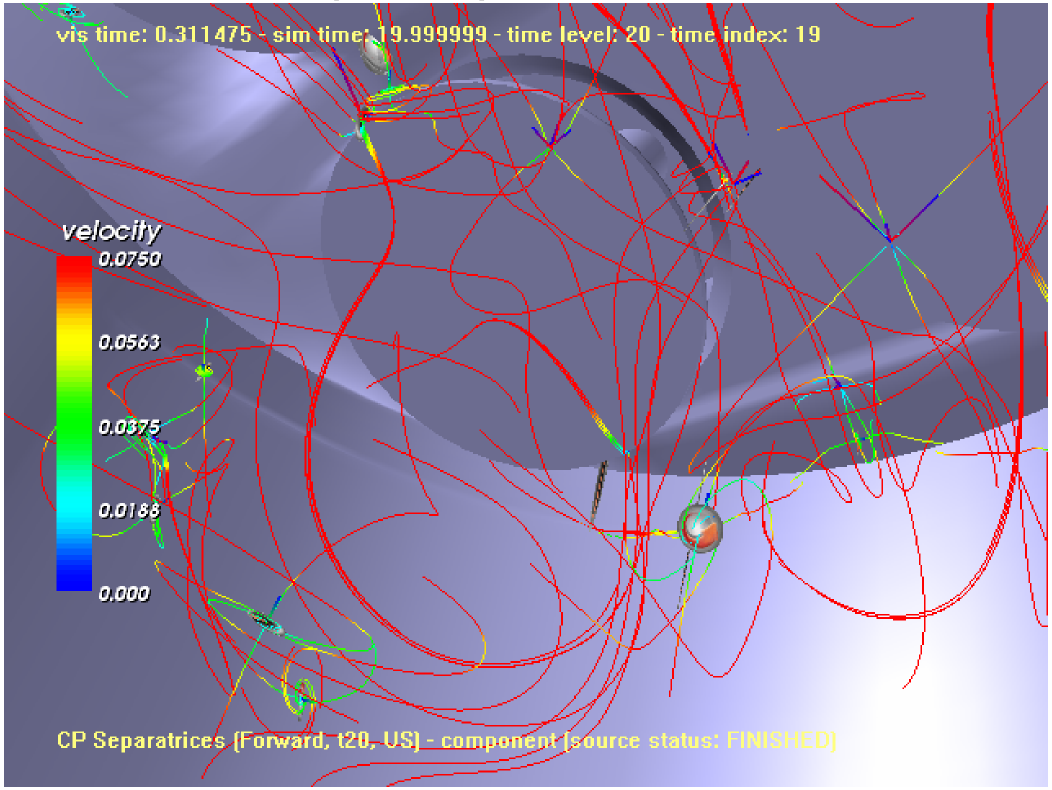
FIRE: Nested OpenMP improves scalability

Speedup of FIRE	Sun Fire E25K, 72 dual-core UltraSPARC-IV processors		
# Threads	Only outer level	Only inner level	Nested OpenMP
4	---	3.8	---
8	---	7.6	---
16	14.8	14.1	15.4
32	29.6	28.9	30.6
72	56.5	---	67.6
144	---	---	133.3

- How can Nested OpenMP improve the scalability?
  - Some synchronization for output ordering on the higher level
  - OpenMP implementation overhead increases over-linear with the number of threads
  - Dataset might better fit to the number of threads

NestedCP: Parallel Critical Point Extraction

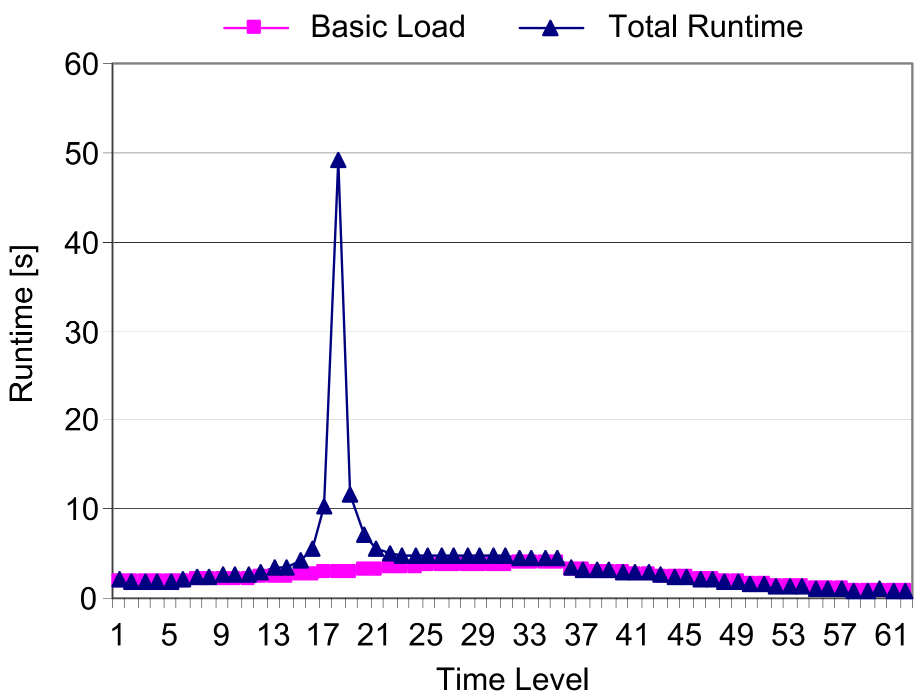
- Virtual reality: Analysis of large-scale flow simulations
  - Feature extraction from raw data
  - Interactive analysis in virtual environment (e.g. a cave)
- Critical point: point in the vector field where the velocity is zero.

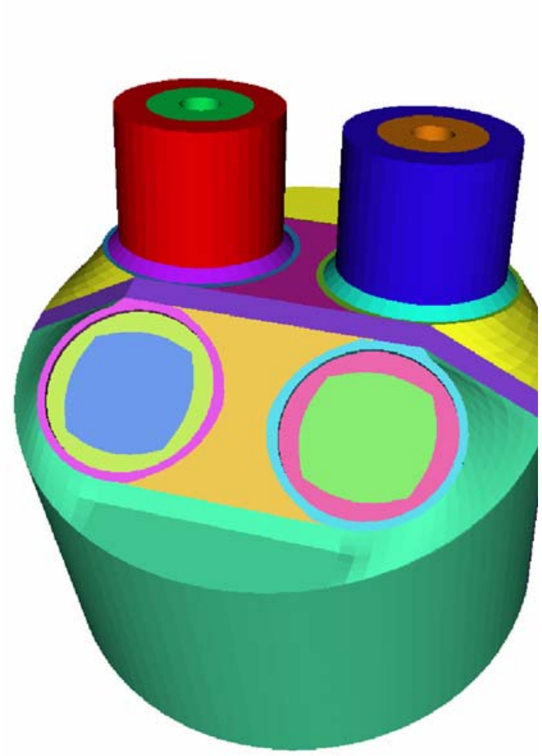


Andreas Gerndt
Virtual Reality Center Aachen

NestedCP: Addressing load imbalance

- Algorithm for critical point extraction:
  - Loop over the time steps of unsteady datasets
  - Loop over the blocks of multi-block datasets
  - Loop checking the cells within the blocks





- The time needed to check different cells may vary considerably!

NestedCP: Addressing load imbalance

- Solution in OpenMP is simple:

```
#pragma omp parallel for num_threads(nTimeThreads) \
                     schedule(dynamic,1)
for (cutT = 1; curT <= maxT; ++curT)
{
#pragma omp parallel for num_threads(nBlockThreads) \
                     schedule(dynamic,1)
   for (curB = 1; curB <= maxB; ++curB)
   {
#pragma omp parallel for num_threads(nCellThreads) \
                     schedule(guided)
      for (curC = 1; curC <= maxC; ++curC)
      {
         findCriticalPoints(curT, curB, curC);
} } }
```

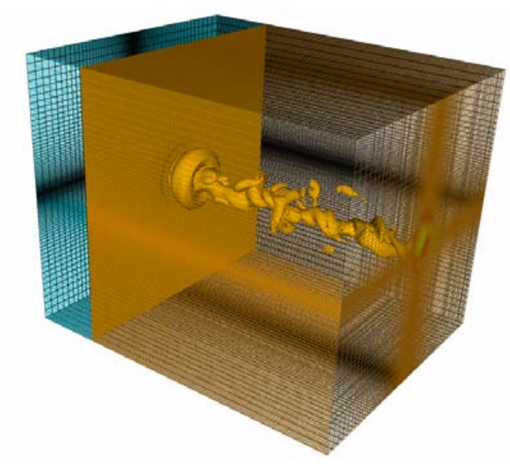
NestedCP: Addressing load imbalance

- The achievable speedup heavily depends on the selected dataset
- No load imbalance → almost perfect scalability can be achieved

- Speedup on Sun Fire E25K, 72 dual-core UltraSPARC-IV processors with 128 threads:
  - Without load-balancing: 10.3
    static scheduling
  - Dynamic / Guided 33.9
    Time = 4, Block = 4, Cell = 32
  - Dynamic / Guided (Sun extension) 55.3
    Weight factor = 20

NestedCP: Addressing load imbalance

- High speed-up
  - Nearly optimal mainly by time-level parallelization
    - T32, B1, C4: 125.30
  - Block sizes considerably larger
  - Dataset shows good balancing
  - Static case only 8% worse



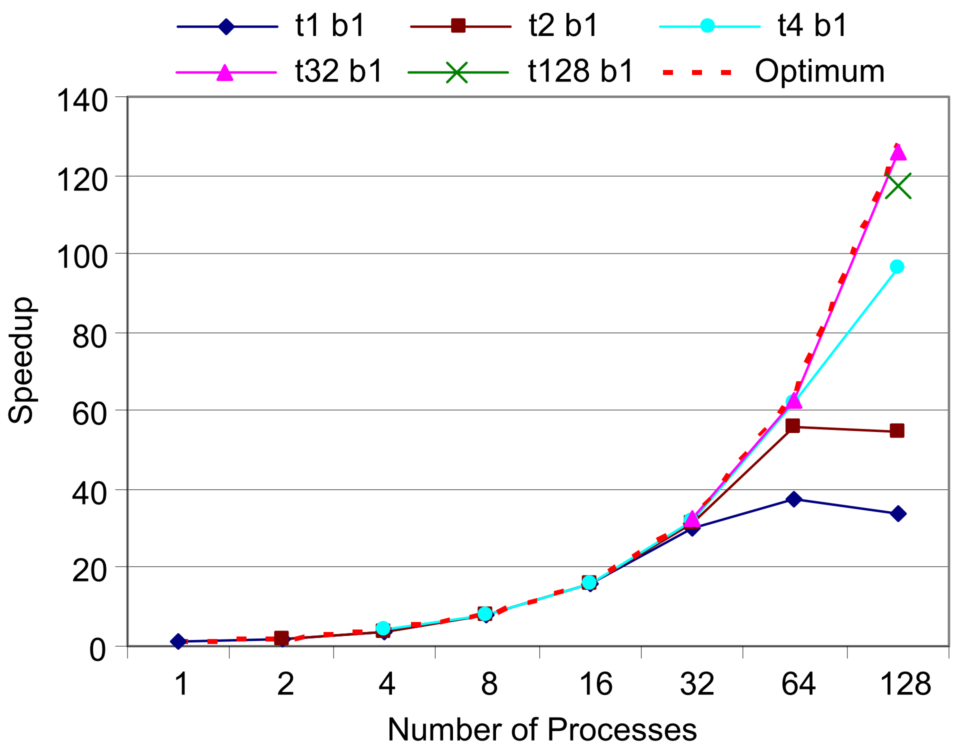
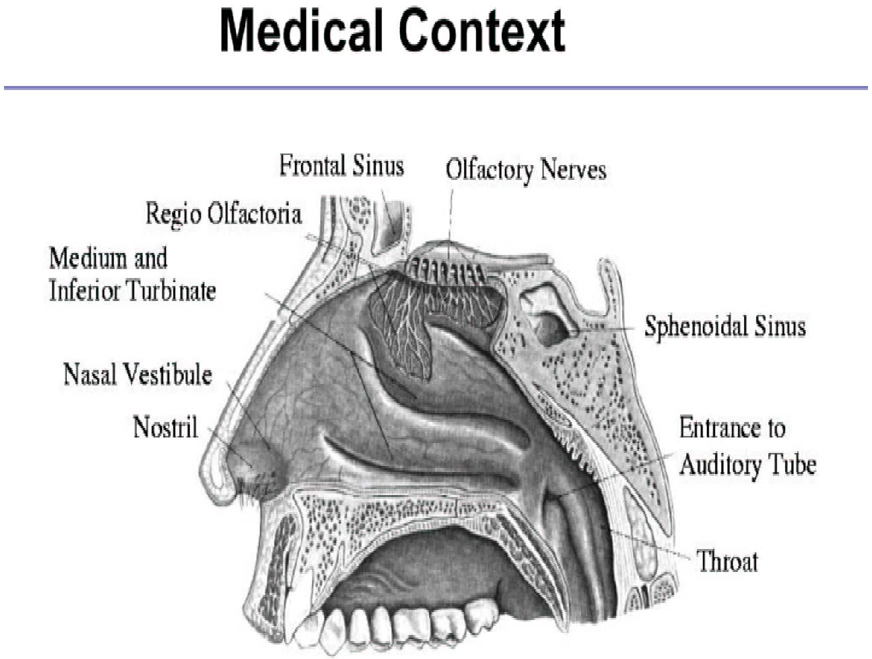


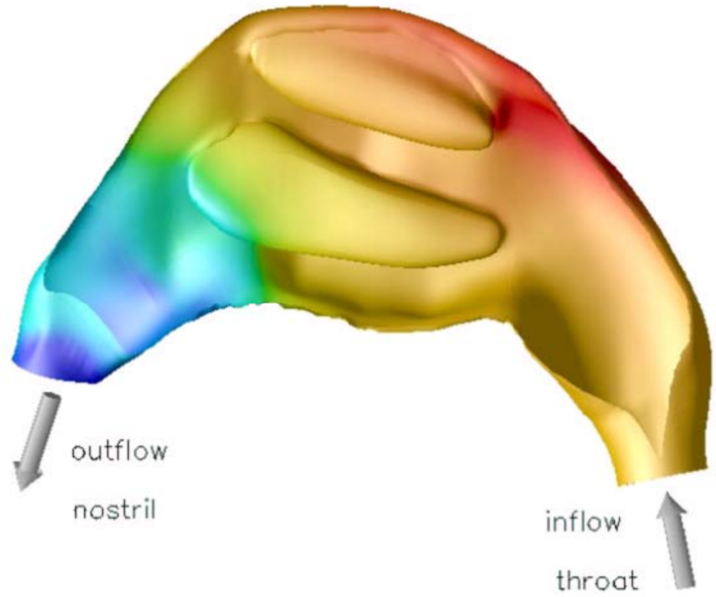
Fig.: Speed-up; Shock Dataset; automatic load balancing

TFS - nested parallelization with OpenMP

- ***TFS*** models human nasal flow for computer aided surgery

Ingolf Hörschler, Aerodynamic Institute, RWTH Aachen
Steve Johnson, Cos Ierotheou, Parallel Software Products

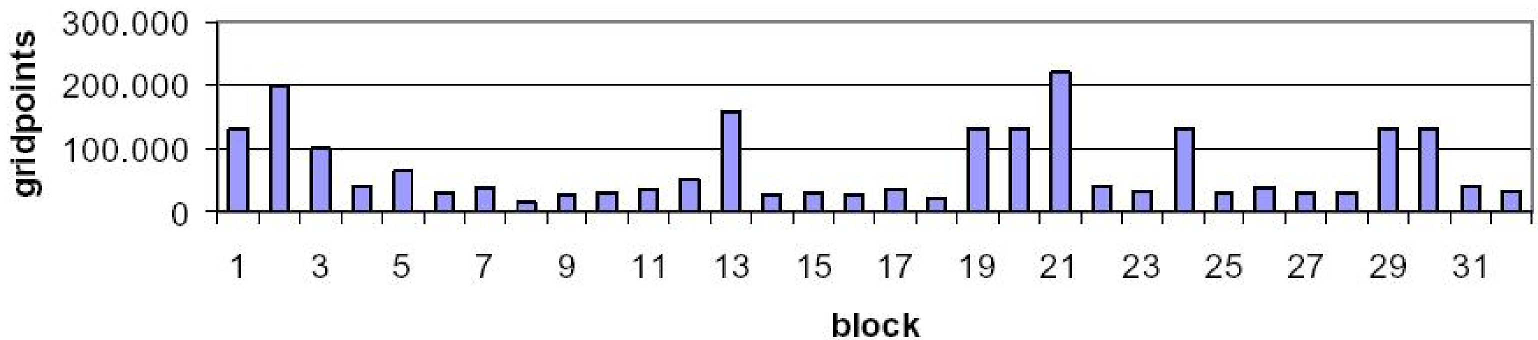




- Require an efficient *OpenMP* version of ***TFS*** code which contains ~20000 lines in 141 routines with 583 DO loops

TFS

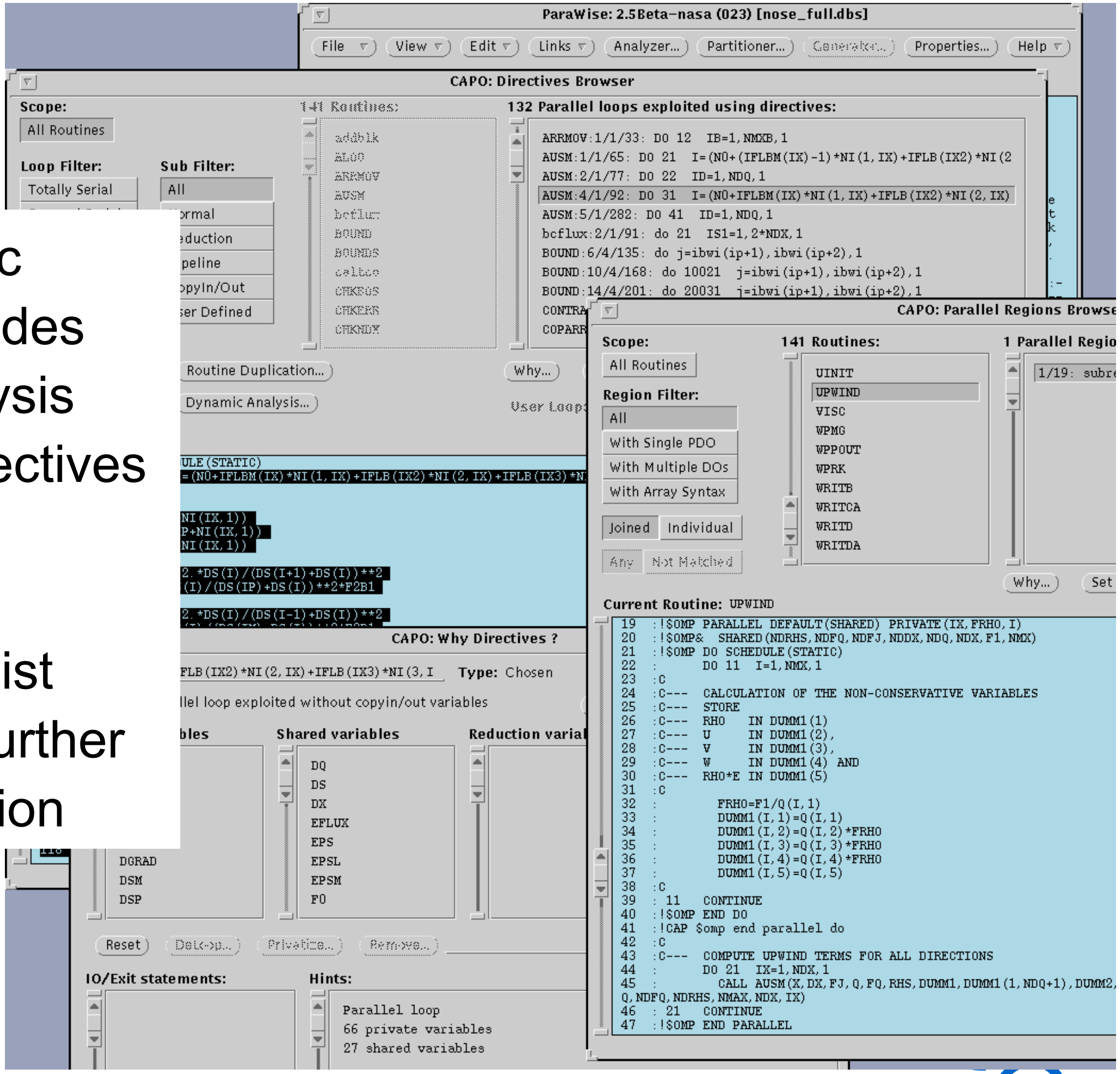
- **TFS** has been tuned for vectorization (performs excellent on NEC SX8)
- Loop level parallelization delivers speedup of 5-6 using 8 threads (US IV)
- MPI on block level would be laborious because of complex geometry
- Also, blocks differ heavily in size => load imbalance



- ParaWiseToolkit for (semi-) automatic parallelization of Fortran codes
  - Global static program analysis, generation of OpenMP directives
  - Profile feedback,
  - Expert system (GUI) to assist user for further improvement of parallelization

ParaWise/CAPO by Parallel Software Products

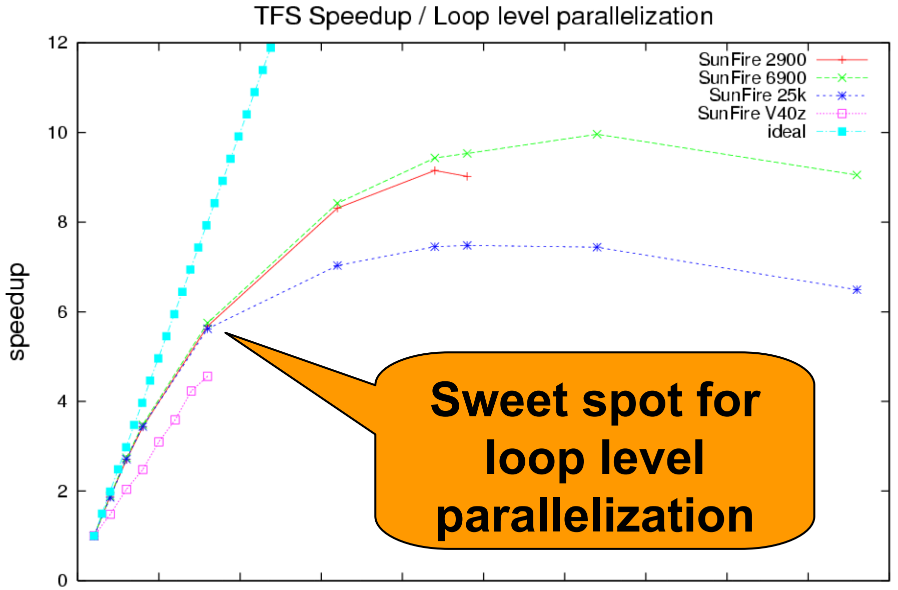
www.parallelsp.com

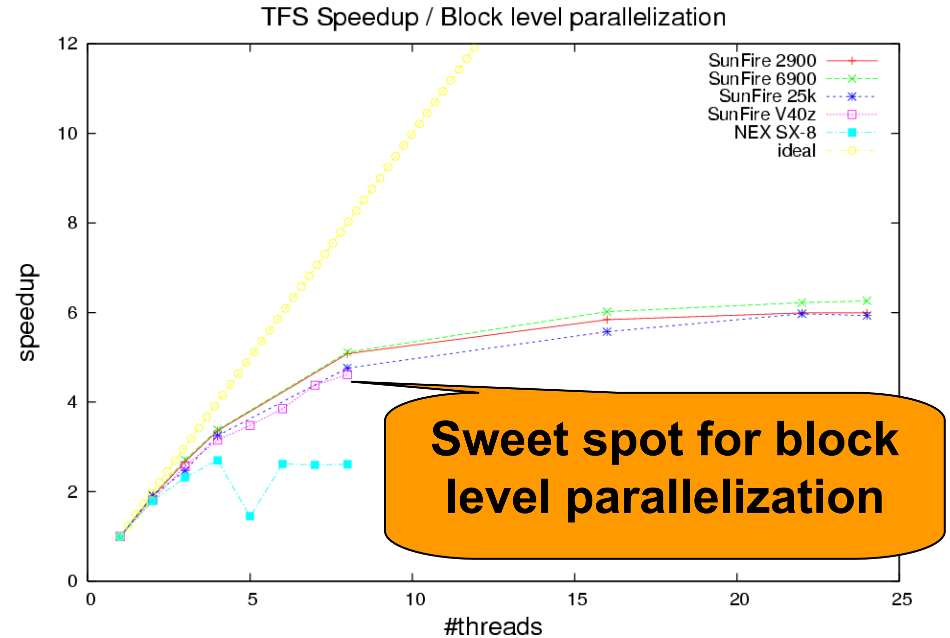


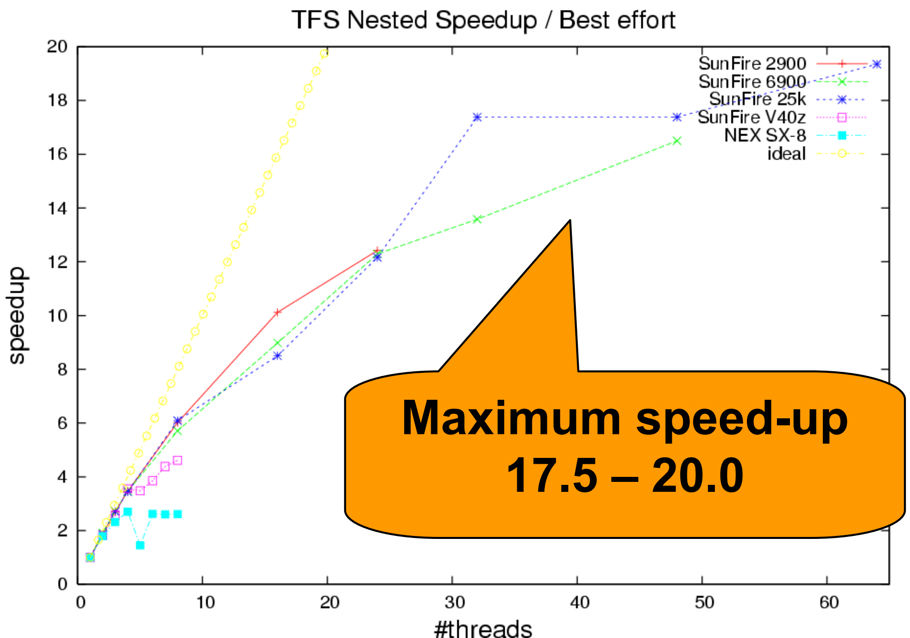
- Toolkit for (semi-) automatic parallelization of Fortran codes
- Global static program analysis
- Generation of OpenMP directives (CAPO, NASA Ames)
- Runtime analysis
- Expert system (GUI) to assist user to give feed-back for further improvement of parallelization

RWTH Center for Computing and Communication

TFS Speedup



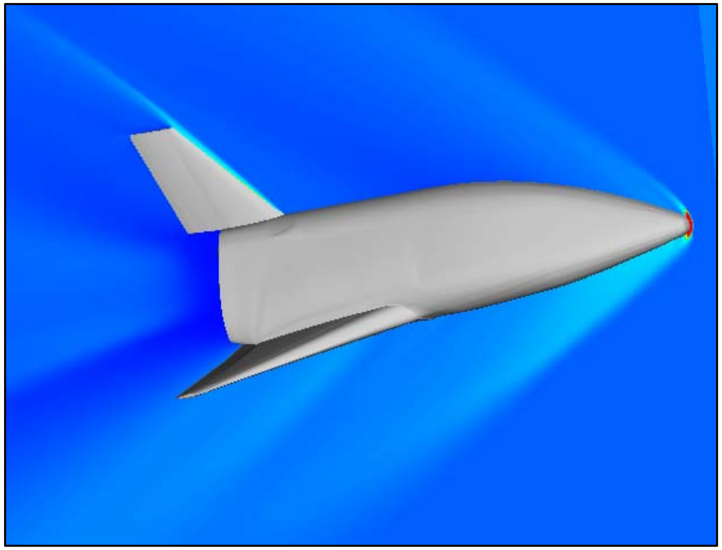




- **experimental threading library from Sun (maintain thread affinity) +**
- **processor binding +**
- **data migration (Solaris) =>**
- **increase speedup with 64 threads from 20 to 25 !**

Navier-Stokes Solver FLOWer

- Navier-Stokes-Solver **FLOWer** (DLR=German Aerospace Center)
- PHOENIX, a small scale prototype
  of the space launch vehicle HOPPER
  designed to take off horizontally and
  glide back to earth after placing its cargo in orbit.

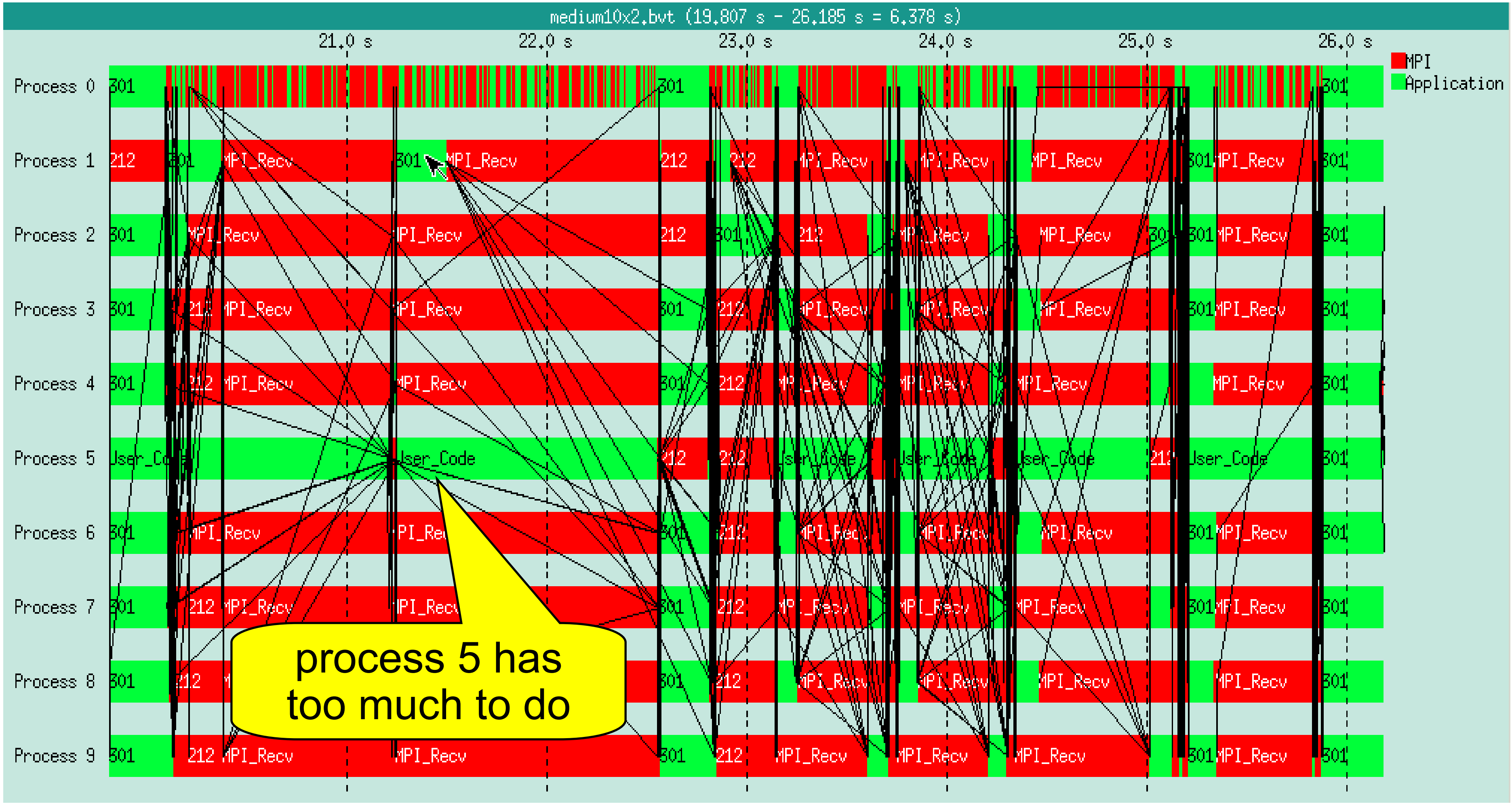


Birgit Reinartz, Michael Hesse, Laboratory of Mechanics, RWTH Aachen University

MPI + autoparallelization => hybrid

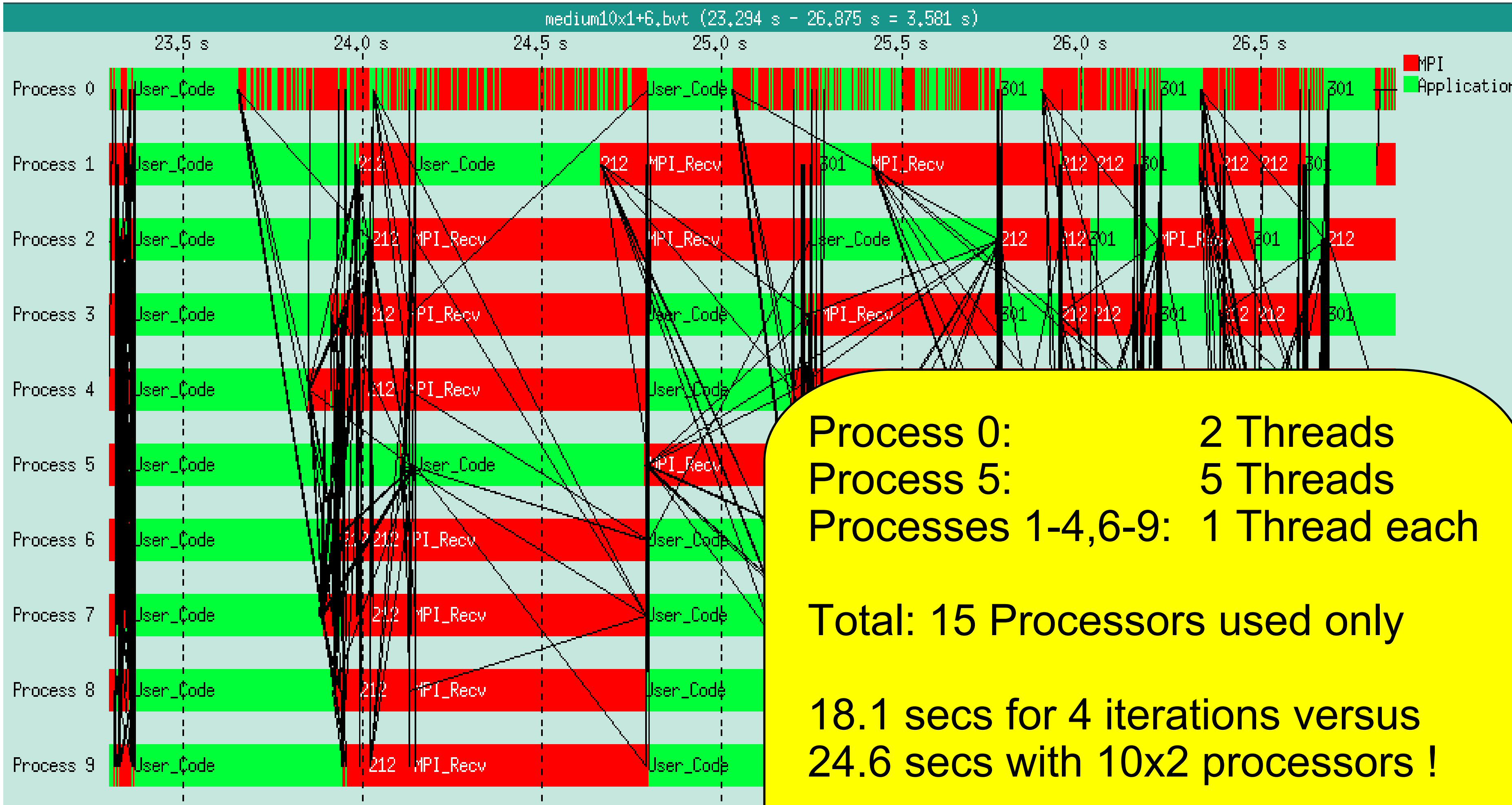
DTB Library to automatically adjust number of threads to improve load balance of MPI version.

FLOWer - Vampir Timeline Display (zoomed) 10 x 2 Processors



RWTH Center for Computing and Communication

FLOWer - Manual Load Balancing
10x1+5 Processors



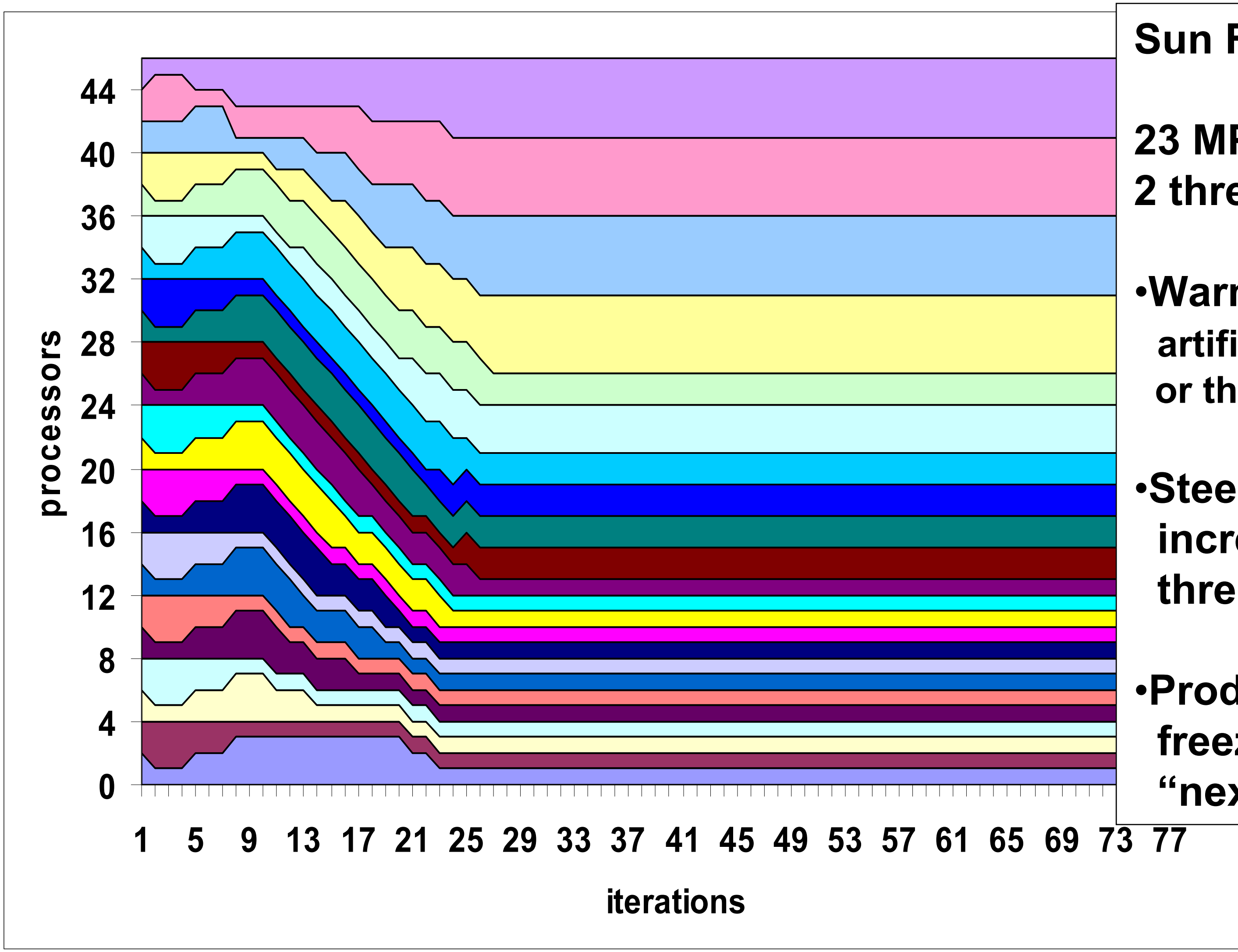
Process 0: 2 Threads
Process 5: 5 Threads
Processes 1-4,6-9: 1 Thread each

Total: 15 Processors used only

18.1 secs for 4 iterations versus
24.6 secs with 10x2 processors !

1.8 times higher efficiency

FLOWer - Dynamic Thread Balancing 23x2 Processors



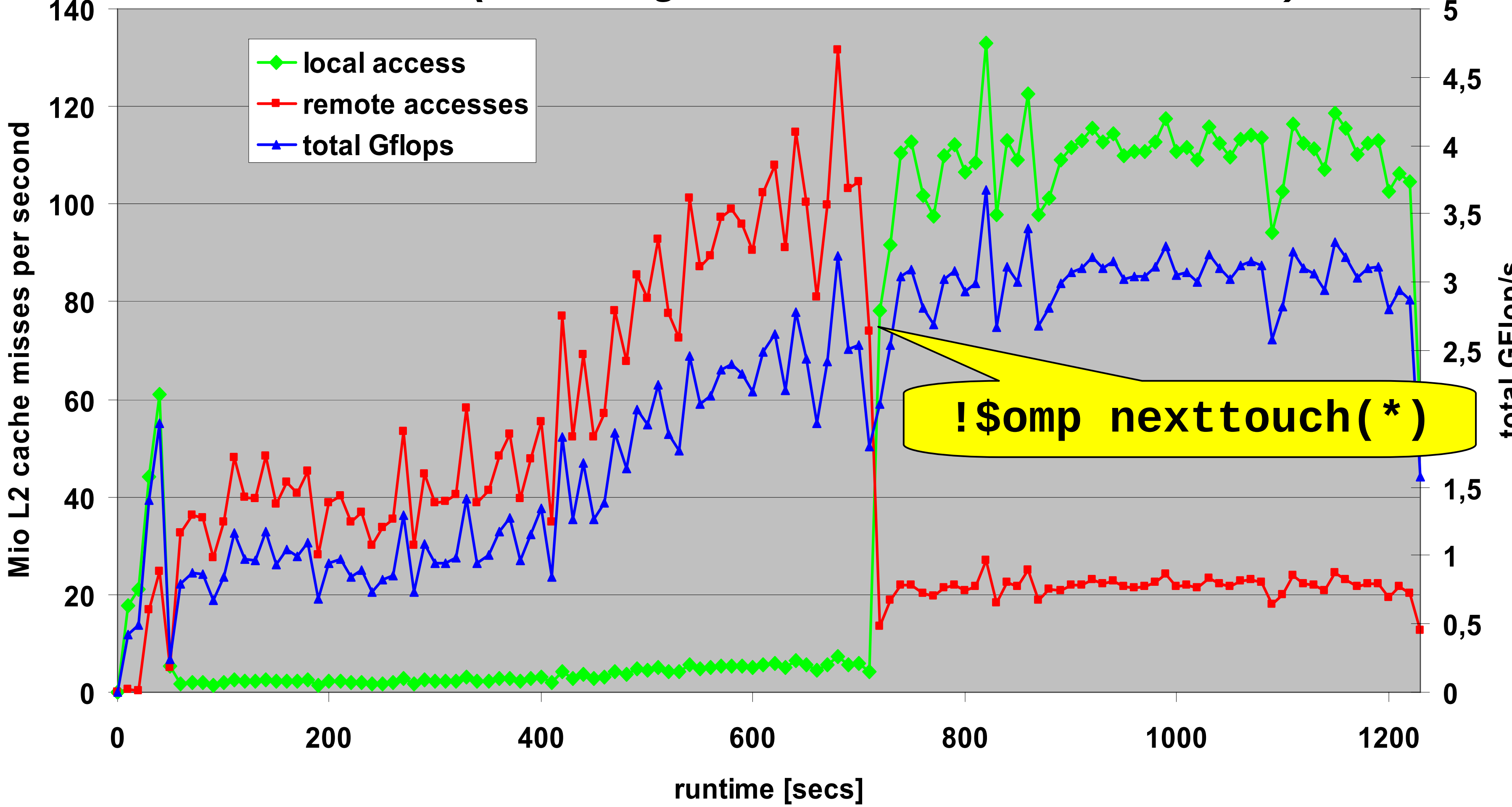
Sun Fire E25K

23 MPI procs start with 2 threads each.

- **Warm-up phase (1-12)** artificially vary number or threads per process
- **Steering phase (13-30)** increase number of threads of busy procs
- **Production phase (31-)** freeze thread numbers "nexttouch"

FLOWer - Dynamic Thread Balancing
23x2 Processors

Sun Fire 25K, ~ 65 Mflop/s per thread = 3% of peak performance
(due to high MPI communication overhead)



RWTH
Center for Computing and Communication

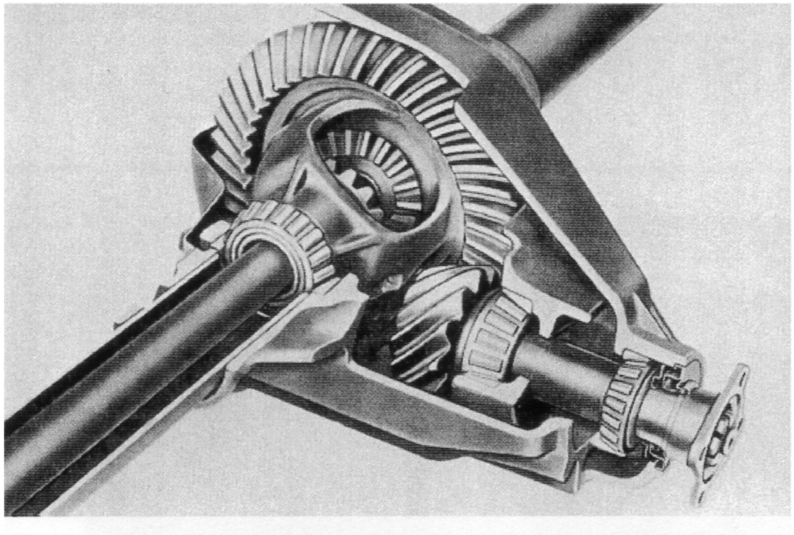
Overview

- Loop-Level Parallelization in Fortran
  - Autoscoping
  - Combining Autoparallelization, OpenMP, Sun Performance Library
  - Pushing Loop-Level Parallelization to the Limit
- C++ and OpenMP
  - DROPS
  - Realtime FEM for VR
- Nested Parallelization
  - Pattern Recognition
  - Critical Points
  - TFS parallelized with Parawise by PSP
  - Dynamic Thread Balancing for MPI+OMP
- **OpenMP Tools / OpenMP on Windows**
- CMP / CMT

FVA346 - Contact Analysis of Bevel Gears

Bevel Gear Pair

Differential Gear



Laboratory for Machine Tools and Production Engineering, RWTH Aachen University

Tuning and Parallelization of a Fortran90 Application for Windows

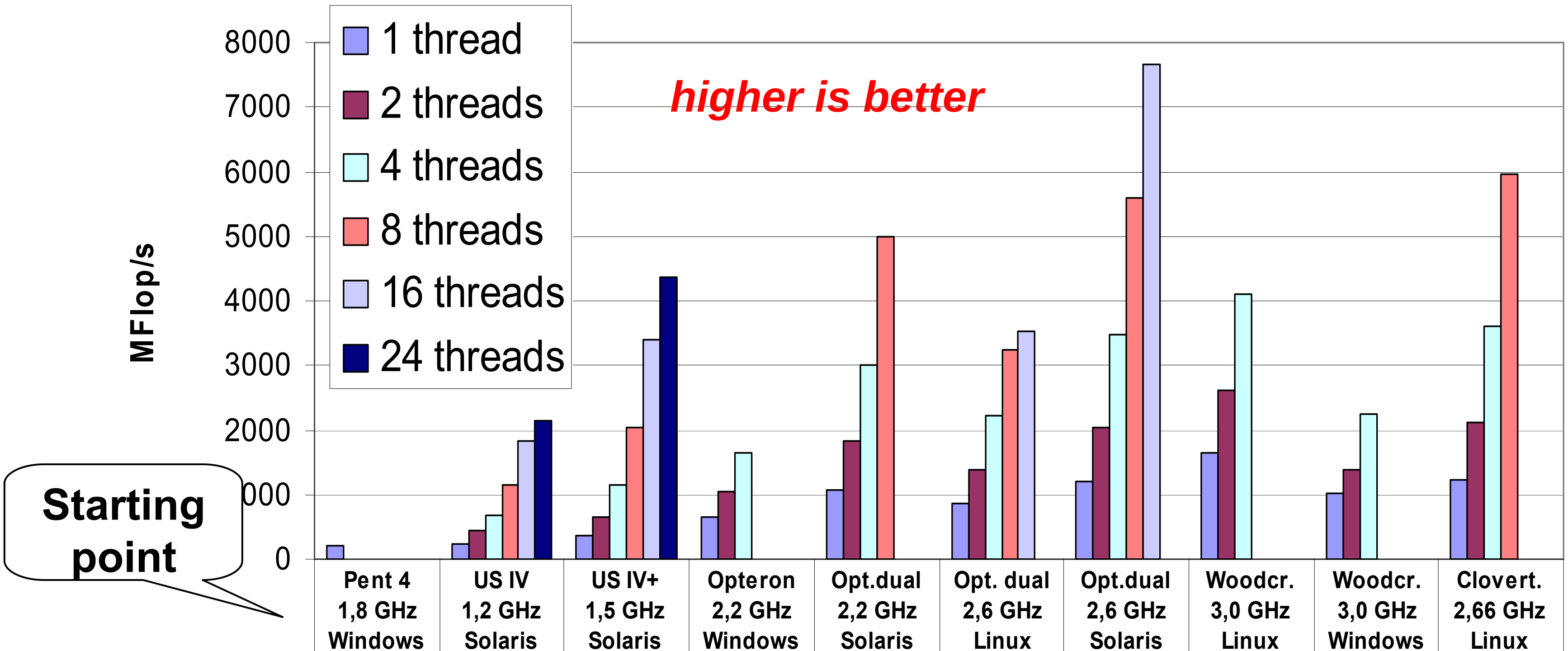
Target

- Pentium/Windows/Intel → Opteron/Windows/Intel + OpenMP + tuning

Procedure

- UltraSPARC IV / Solaris / SunStudio
- Etnus **TotalView on** : Porting (~ 2x2 hours)
- Simulog **Foresys**: Fortran77 → Fortran 90 (90,000 lines of code)
- **Sun Analyzer**: Runtime analysis
- OpenMP-Parallelization (~ 2 days incl. serial tuning, 5 PR, 70 directives)
- Intel Compiler on Linux
- Intel **ThreadChecker**: Verification of OpenMP version
- Opteron/Windows/Intel

Performance (Mflop/s)

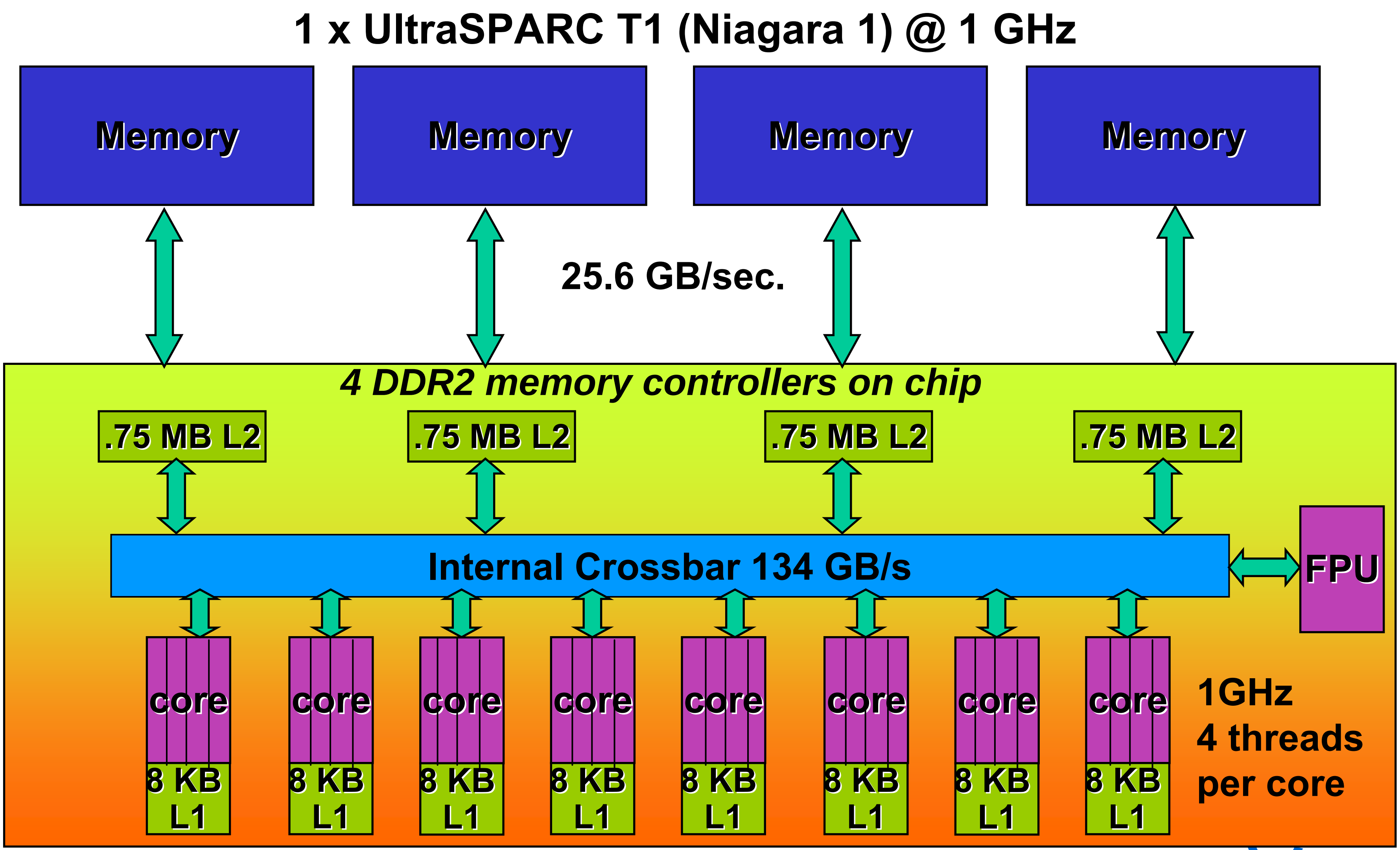


	Pent 4 1,8 GHz Windows	US IV 1,2 GHz Solaris	US IV+ 1,5 GHz Solaris	Opteron 2,2 GHz Windows	Opt.dual 2,2 GHz Solaris	Opt. dual 2,6 GHz Linux	Opt.dual 2,6 GHz Solaris	Woodcr. 3,0 GHz Linux	Woodcr. 3,0 GHz Windows	Clovert. 2,66 GHz Linux
1 thread	220	224	368	647	1072	861	1195	1649	1032	1237
2 threads		444	645	1033	1824	1382	2052	2602	1388	2125
4 threads		681	1152	1644	3011	2216	3482	4112	2255	3613
8 threads		1148	2047		4996	3246	5587			5950
16 threads		1829	3398			3534	7658			
24 threads		2144	4369							

Overview

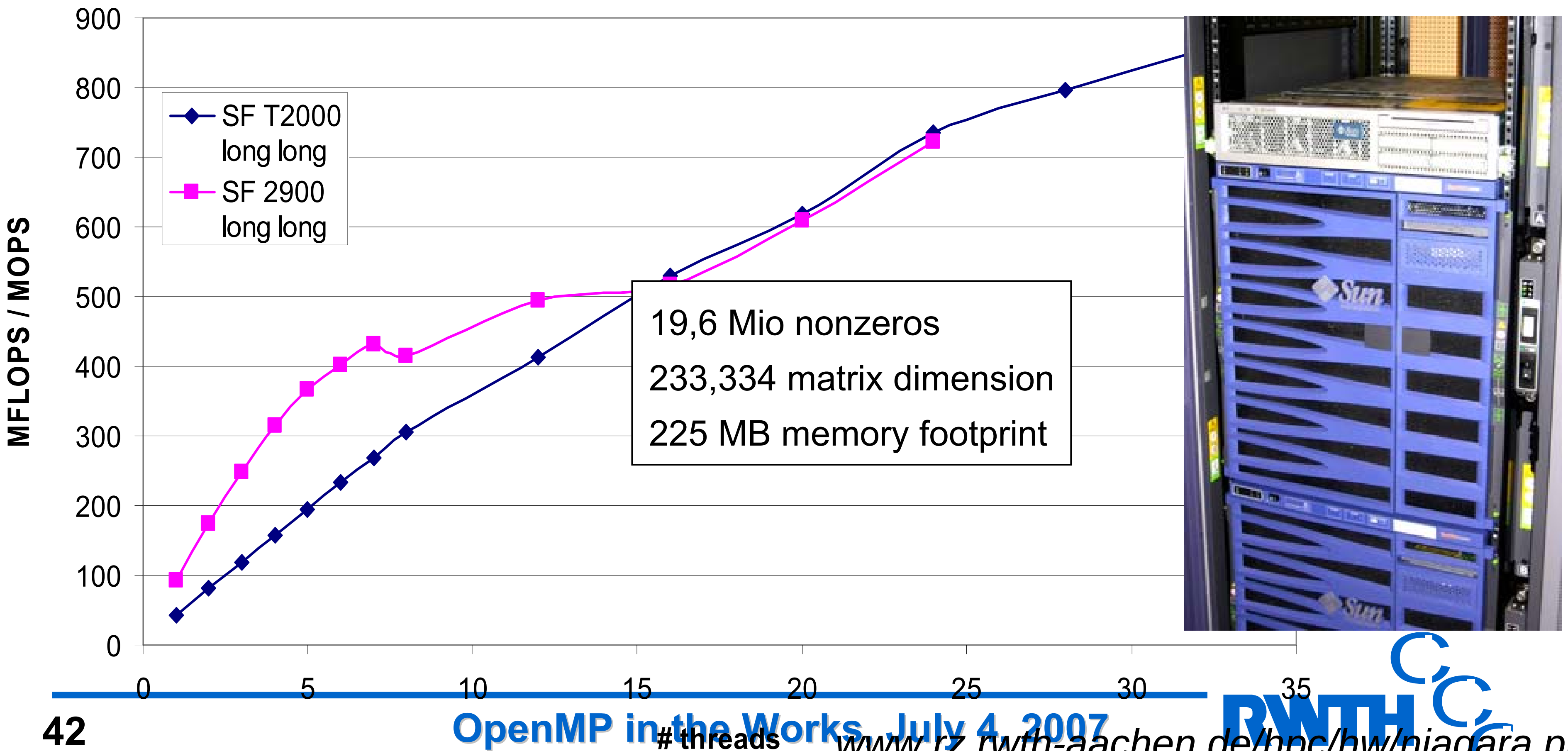
- Loop-Level Parallelization in Fortran
  - Autoscoping
  - Combining Autoparallelization, OpenMP, Sun Performance Library
  - Pushing Loop-Level Parallelization to the Limit
- C++ and OpenMP
  - DROPS
  - Realtime FEM for VR
- Nested Parallelization
  - Pattern Recognition
  - Critical Points
  - TFS parallelized with Parawise by PSP
  - Dynamic Thread Balancing for MPI+OMP
- OpenMP Tools / OpenMP on Windows
- **CMP / CMT**

Sun Fire T2000 – Eight Cores x Four Threads

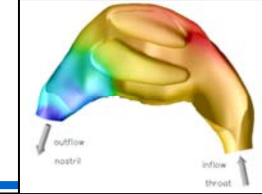


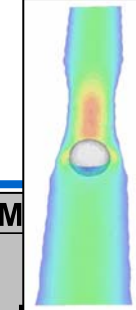
Sparse "Pseudo"-Matrix Vector Multiplication (long long int)

Top: Sunfire T2000 (1 x 8-core multithreaded Niagara, 1 GHz)
Bottom: Sunfire 2900 (12 x 2-core Ultrasparc IV, 1,2 GHz)



www.rz.rwth-aachen.de/hpc/hw/niagara.ph





			BevelGears			ThermoFlow			TFS			DROPS (sM		
	OpenMP Threads	Sockets-Chips-Cores-Threads	Runtime [sec]	Speedup	Eff.	Runtime [sec]	Speedup	Eff.	Runtime [sec]	Speedup	Eff.	Speed [Mflop/s]	Speedup	Eff.
USIV 1.2	1	1-1-1-1	276.33	1.00	100	66.4	1.0	100	33.6	1.0	100	141.0	1.00	100
	2	1-1-2-1	161.62	1.71	85	42.8	1.55	78	18.8	1.79	90	274.8	1.95	97
	2	2-1-1-1	162.95	1.70	85	42.2	1.57	79	18.9	1.78	89	243.7	1.73	86
	4	2-1-2-1	90.87	3.04	76	28.9	2.30	58	11.4	2.95	74	423.8	3.01	75
Opteron 2.2	1	1-1-1-1	57.96	1.00	100	29.2	1.0	100	14.3	1.0	100	371.8	1.00	100
	2	1-1-2-1	34.91	1.66	83	20.7	1.41	71	11.6	1.23	62	643.5	1.73	87
	2	2-1-1-1	34.79	1.67	83	18.5	1.58	79	11.6	1.23	62	721.6	1.94	97
	4	2-1-2-1	20.25	2.86	72	13.6	2.15	54	8.7	1.64	41	1200.0	3.23	81
Wood-crest 3.0	1	1-1-1-1	36.02	1.00	100	20.1	1.0	100	10.0	1.0	100	557.4	1.00	100
	2	1-1-2-1	21.73	1.66	83	16.1	1.25	63	8.0	1.25	63	643.5	1.15	58
	2	2-1-1-1	21.71	1.66	83	15.0	1.34	67	6.0	1.67	84	872.9	1.57	78
	4	2-1-2-1	14.64	2.46	62	13.0	1.55	39	5.0	2.0	50	951.3	1.71	43
Clover town 2.66	1	1-1-1-1	48.10	1.00	100	23.1	1.00	100	11.10	1.00	100	495.2	1.00	100
	2	1-1-2-1	28.10	1.71	86	19.0	1.22	61	9.10	1.22	61	563.9	1.14	57
	2	1-2-1-1	28.10	1.71	86	17.0	1.36	68	7.00	1.59	79	764.3	1.54	77
	2	2-1-1-1	28.00	1.72	86	17.1	1.35	68	8.00	1.39	69	622.5	1.26	63
	4	1-2-2-1	16.10	2.99	75	14.1	1.64	41	6.00	1.85	46	902.9	1.82	46
	4	2-1-2-1	16.10	2.99	75	15.1	1.53	38	7.10	1.56	39	632.6	1.28	32
	4	2-2-1-1	16.47	2.92	73	13.1	1.76	44	4.10	2.71	68	807.9	1.63	41
	8	2-2-2-1	10.00	4.81	60	13.0	1.78	22	4.10	2.71	34	913.2	1.84	23

Parallel Efficiency >80%
Parallel Efficiency 60-80%
Parallel Efficiency 40-60%
Parallel Efficiency <40%

Conclusion

- For our user community – engineers and natural scientists - OpenMP frequently is an interesting alternative, as MPI parallelization would cause much more work.
- Anyway, quite a few MPI codes are running at our site, most of them have been "imported".
- Tools for performance analysis and verfication are critical ingredients of the programming development environment
- OpenMP is useful for multicore architectures.
- But the memory wall will be hitting us even more in the future => CMT.
- OpenMP 3.0 tasking concept will greatly enhance its usability.