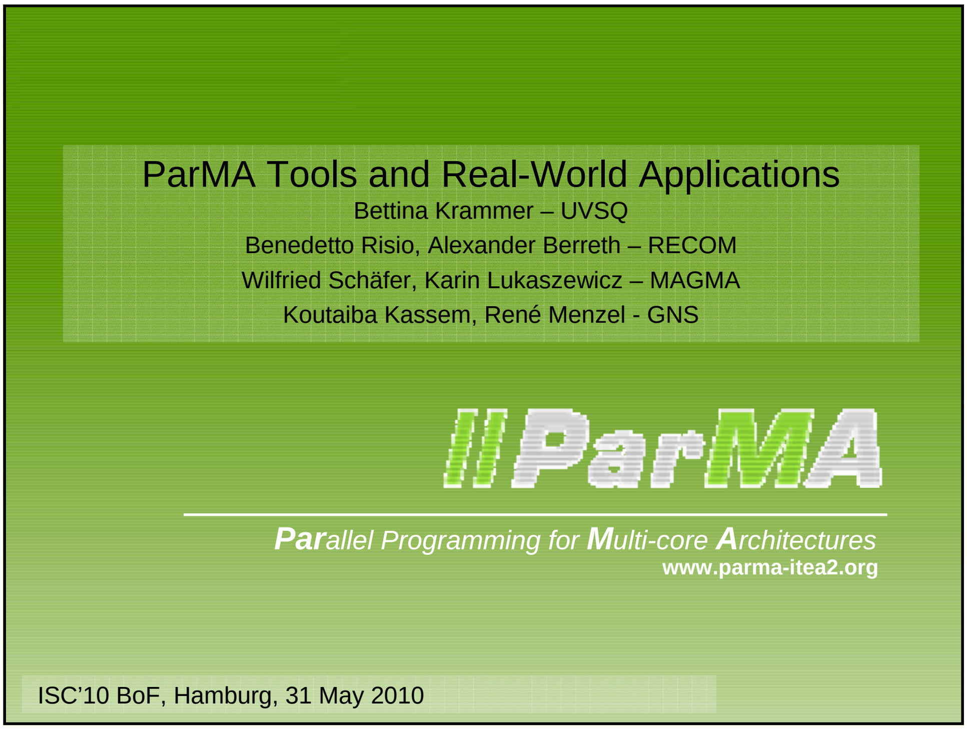
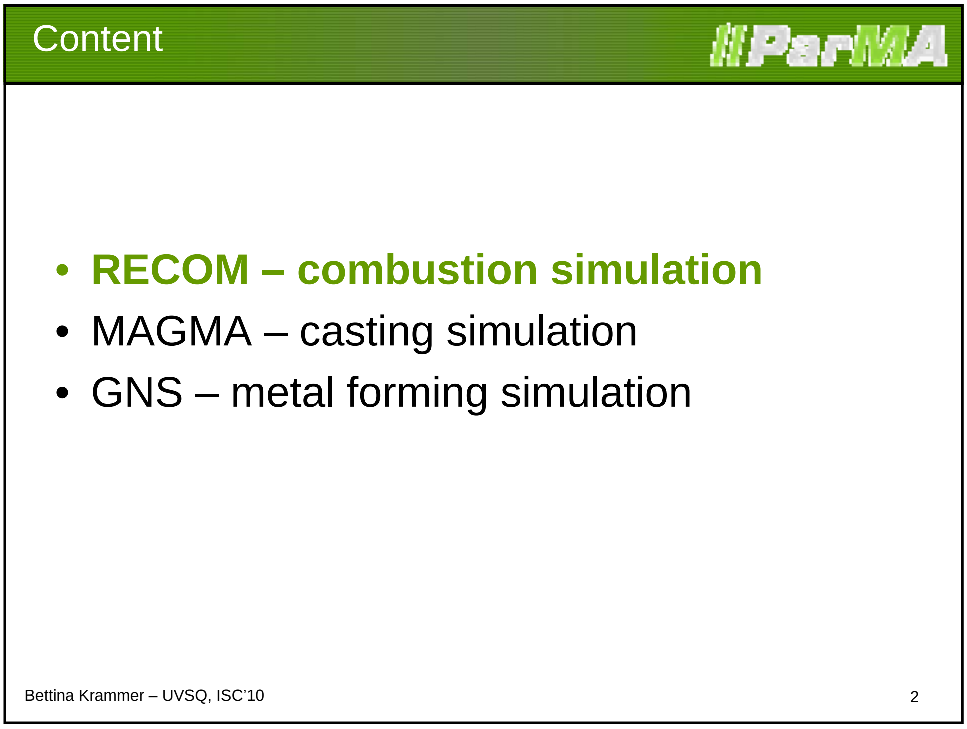


ParMA Tools and Real-World Applications

Bettina Krammer – UVSQ
Benedetto Risio, Alexander Berreth – RECOM
Wilfried Schäfer, Karin Lukaszewicz – MAGMA
Koutaiba Kassem, René Menzel - GNS

***Par**allel Programming for **M**ulti-core **A**rchitectures*
www.parma-itea2.org

ISC'10 BoF, Hamburg, 31 May 2010



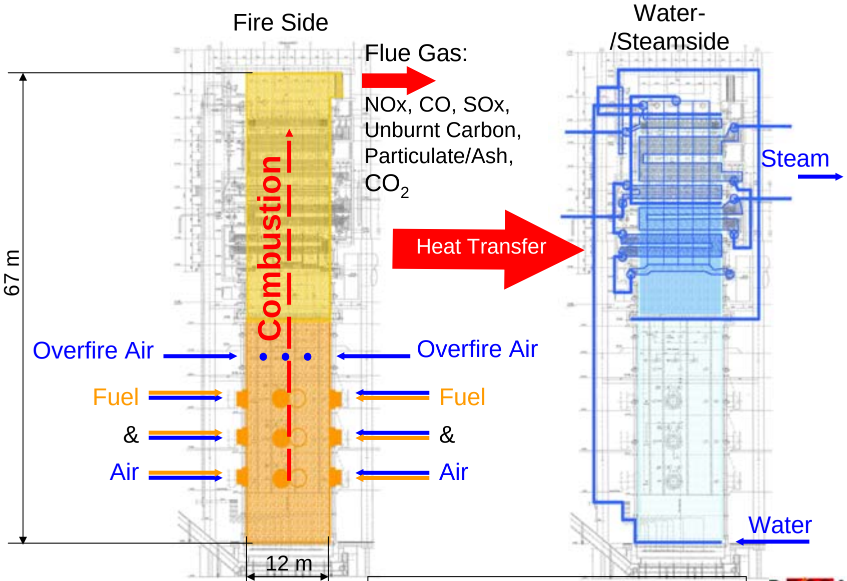
Content

- **RECOM – combustion simulation**
- MAGMA – casting simulation
- GNS – metal forming simulation

RECOM Services GmbH – Business Area

Computer-Aided combustion optimisation in fossil-fueled power stations

Schematic of power generation in fossil fueled power stations



Compute-intensive simulation!

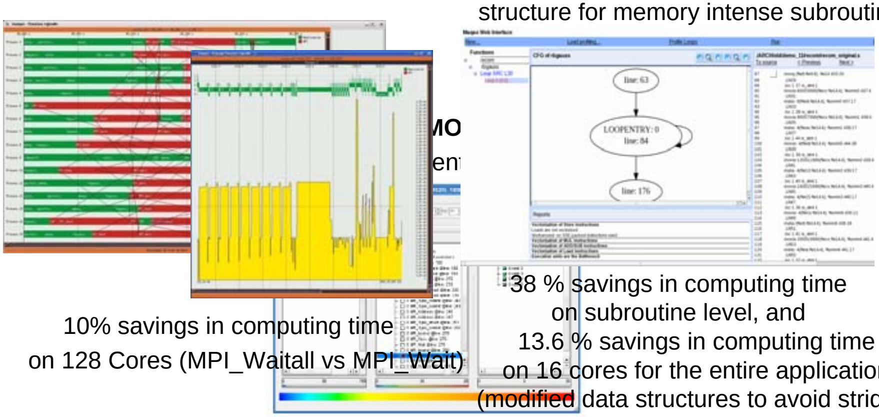
Results of the optimization on Xeon

VAMPIR / SCALASCA

Analysis of MPI-parallel execution & communication

MAQAO

Analysis of memory access and execution structure for memory intense subroutines



10% savings in computing time on 128 Cores (MPI_Waitall vs MPI_Wait)

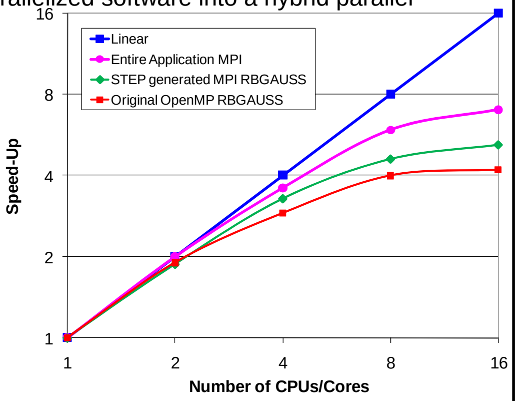
38 % savings in computing time on subroutine level, and 13.6 % savings in computing time on 16 cores for the entire application (modified data structures to avoid stride 2 access)

Data type conversion error identified and removed

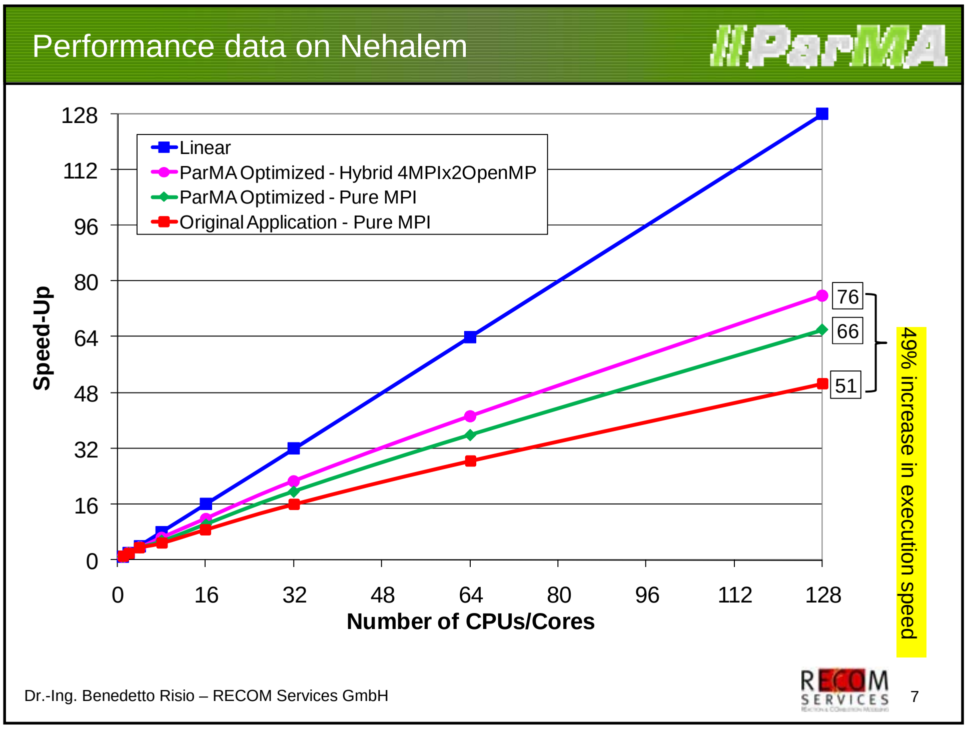
Experimentation with STEP on Xeon

- **STEP:**
  Automatic conversion of OpenMP-parallelized software into a hybrid parallel (OpenMP and MPI) version.

- **Target of the experimentation:**
  Demonstrate the potential of STEP for the complex RECOM-AIOLOS Software package by automatically converting the RBGAUSS solver

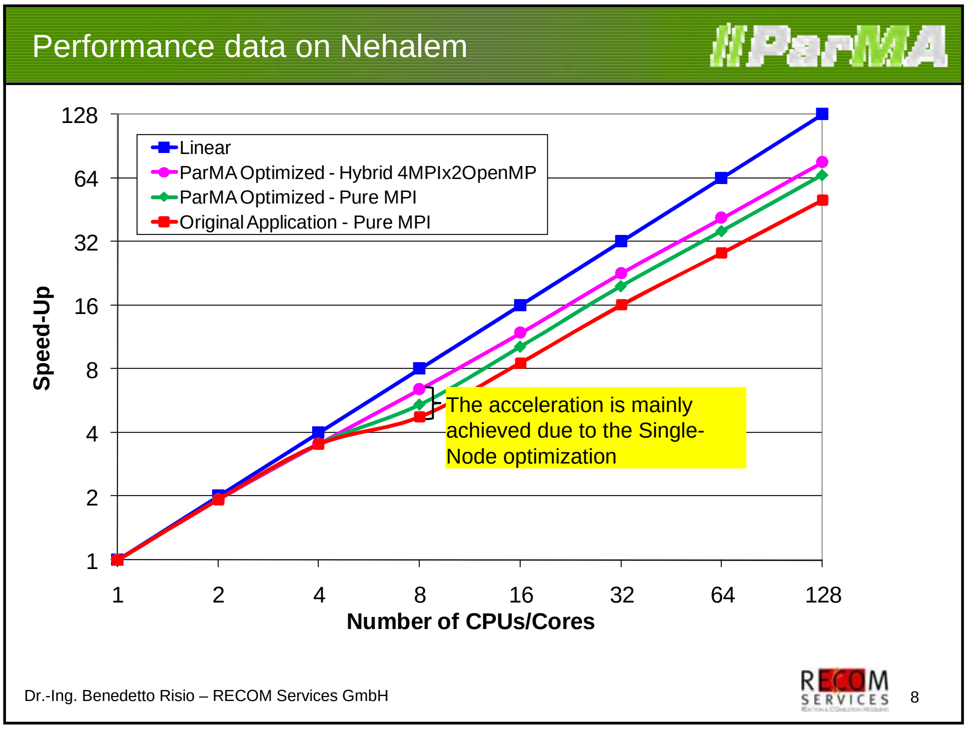


- **Results of the experimentation:**
  STEP successfully converted an OpenMP-parallelized solver into an MPI-parallel version and achieved comparable parallel performance.
  The STEP concept represents a useful approach to decrease development time and costs for MPI-programming.



Performance data on Nehalem

49% increase in execution speed



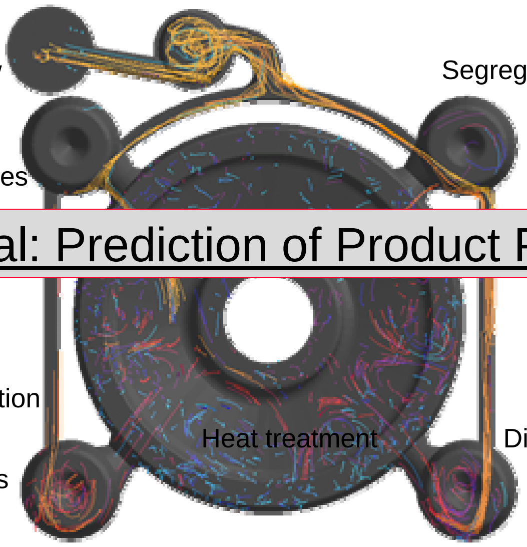
Performance data on Nehalem

The acceleration is mainly achieved due to the Single-Node optimization

Content

- RECOM – combustion simulation
- **MAGMA – casting simulation**
- GNS – metal forming simulation

MAGMA - Casting
the most complex manufacturing process



Flow

Heat flow

Cooling

Gases

Convection

Segregation

Chemistry

Goal: Prediction of Product Properties

Machining

Solidification

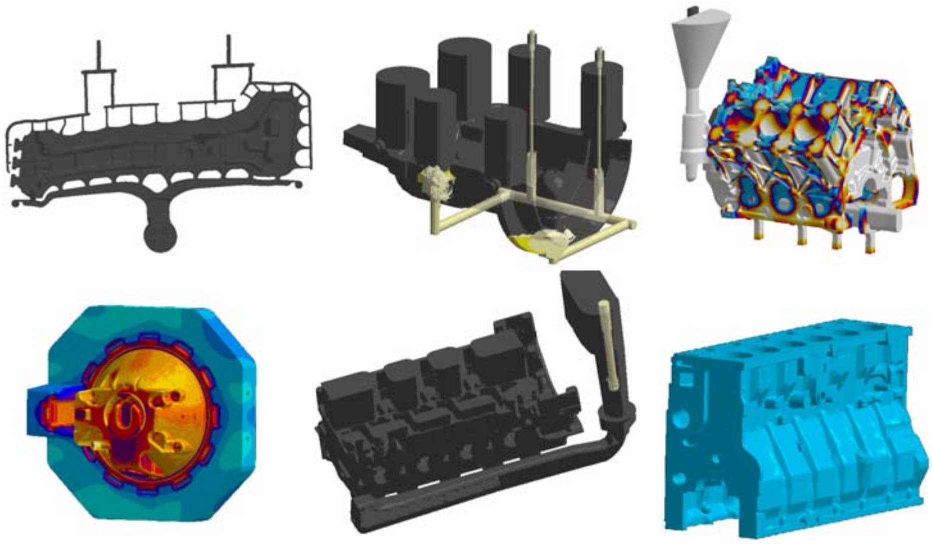
Stresses

Heat treatment

Metallurgy

Distortion

Casting process simulation a core innovation



MAGMASOFT® Optimisation of different modules

- **MAGMAfill**
  - Optimisation of the MPI communication with Scalasca
  - Replace collective by asynchronous communication
  - Overall MPI runtime cut by 3200 seconds (almost by 50%)
  - Overall runtime of the complete simulation reduced by 23%.
- **MAGMAsolid**
  - **Integration of a new linear equation solver and optimisation with Scalasca**
  - **Optimisation of the CG-Solver with MAQAO**
- **MAGMAstress**
  - Optimisation of the communication with Scalasca
  - MPI time approx. reduced by 50%

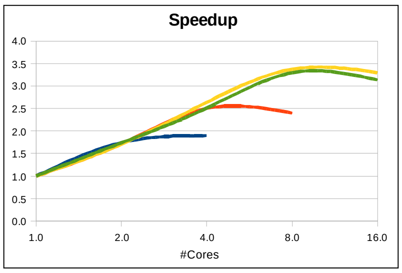
MAGMAsolid
Starting Point: SMP and MPI version

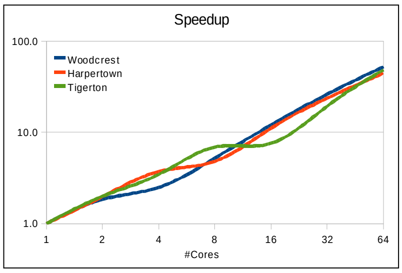
SMP-Version

The solver (ADI, Thomas Algorithm) parallelised with OpenMP does not scale well on a SMP machine and is not suitable for a cluster

MPI-Version

The MPI solver (CGSolver) does scale quite well, but for less than 8 CPU's this solver is much slower than the SMP Version

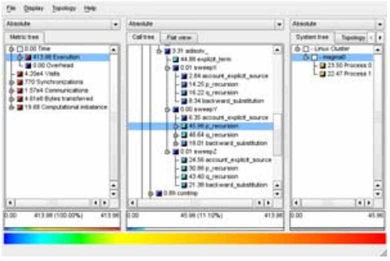




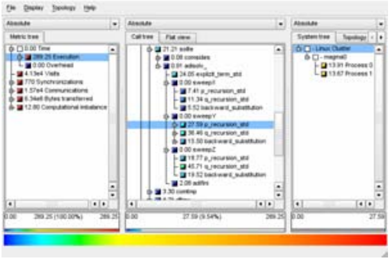
Starting point: The 2 code versions have to be maintained to make customers happy

MAGMAsolid
Optimisation steps

- Integration of a parallel tridiagonal equation solver
- Optimisation with Scalasca



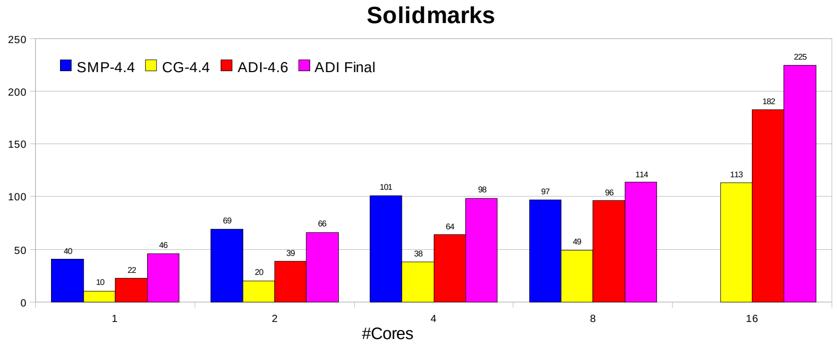
```
0.00 Time
  413.98 Execution
    0.00 Overhead
  4.20e4 Visits
  770 Synchronizations
  1.57e4 Communications
  4.61e8 Bytes transferred
  19.88 Computational imbalance

3.31 adisolv_
  44.88 explicit_term
  0.01 sweepX
    2.84 account_explicit_source
    14.25 p_recursion
    16.22 q_recursion
    8.34 backward_substitution
  0.00 sweepY
    6.35 account_explicit_source
    45.98 p_recursion
    46.64 q_recursion
    18.01 backward_substitution
  0.01 sweepZ
    24.56 account_explicit_source
    30.86 p_recursion
    43.40 q_recursion
    21.38 backward_substitution
0.89 contrip

- Linux Cluster
  - magma0
    23.50 Process 0
    22.47 Process 1

0.00   413.98 (100.00%)   413.98
0.00   45.98 (11.10%)   413.98
0.00   45.98
```



```
0.00 Time
  289.25 Execution
    0.00 Overhead
  4.13e4 Visits
  770 Synchronizations
  1.57e4 Communications
  6.34e8 Bytes transferred
  12.80 Computational imbalance

21.21 solite
  0.08 consides
  0.01 adisolv_
    24.05 explicit_term_std
    0.00 sweepX
      7.41 p_recursion_std
      11.34 q_recursion_std
      5.52 backward_substitution
    0.00 sweepY
      27.59 p_recursion_std
      36.46 q_recursion_std
      13.50 backward_substitution
    0.00 sweepZ
      18.77 p_recursion_std
      45.71 q_recursion_std
      19.52 backward_substitution
    2.08 adifini
3.30 contrip

- Linux Cluster
  - magma0
    13.91 Process 0
    13.67 Process 1

0.00   289.25 (100.00%)   289.25
0.00   27.59 (9.54%)   289.25
0.00   27.59
```

- Optimisation for 1, 2 and 4 cores
  - Using the sequential solver for one core in all three directions, for two cores in two directions and for four cores in one direction

MAGMAsolid
Performance



- New solver (ADI final) competitive with or better than old solver (SMP)
- CG solver sometimes still needed for stability reasons

MAGMAsolid
Analysis of the CG-Solver with MAQAO

Analysed Functions

- scalprod (Vector product of two float vectors)
- scalpnorm (Calculation of the L2 Norm and the MAX Norm)
- saxpy2 (vec1 = vec1 + const * vect2)
- matvec (Matrix Vector product)

Original Code

```
erg=0D0
do k = anf3,end3
  do j = anf2, end2
    do = anf1, end1
      erg = erg +
              DPROD(vec1(i,j,k),vec2(i,j,k))
    end do
  end do
end do
```

Optimised Code

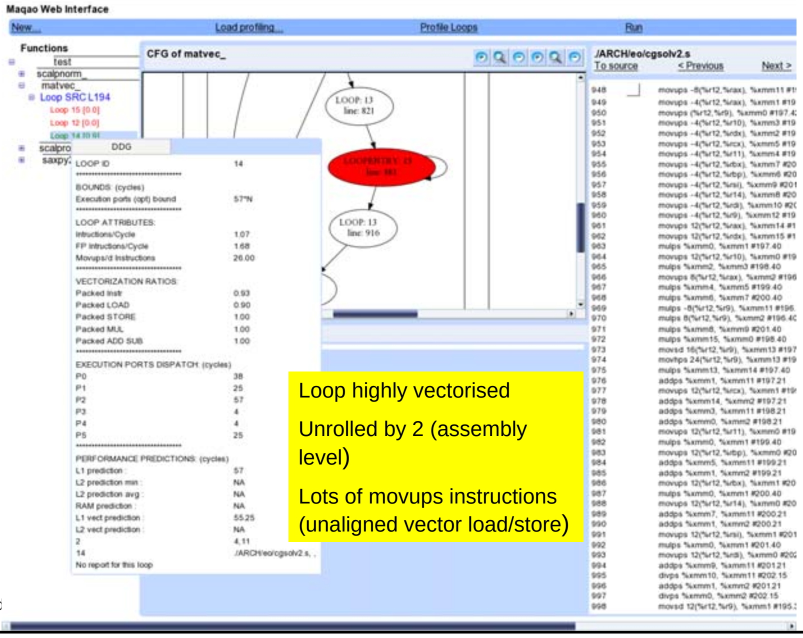
```
end1_1= (anf1+((end1-anf1+1) / 16) * 16) - 1
erg = 0D0
do k = anf3, end3
  do j = anf2, end2
    do i = anf1, end1_1,16
      call MM_PREFETCH (vect1(i+256,j,k), 3)
      call MM_PREFETCH (vect2(i+256,j,k), 3)
      erg = (((((((((((((((erg +
                        DPROD(vec1(i+  0,j,k), vec2(i+  0,j,k)))  +
                        DPROD(vec1(i+  1,j,k), vec2(i+  1,j,k)))  +
                        ...
                        DPROD(vec1(i+15,j,k), vec2(i+15,j,k))
    end do
    do i  =   end1_1+1,   end1
      erg = erg + DPROD(vec1(i,j,k), vec2(i,j,k))
    end do
  end do
end do
```

Prefetching

Loop unrolling (16)

Vector alignment

MAGMAsolid
matvec MAQAO Analysis (original loop)



```
LOOP ID                              14
**********************************
BOUNDS: (cycles)
Execution ports (opt) bound          57*N
**********************************
LOOP ATTRIBUTES:
Intructions/Cycle                    1.07
FP Intructions/Cycle                 1.68
Movups/d Instructions                26.00
**********************************
VECTORIZATION RATIOS:
Packed Instr                         0.93
Packed LOAD                          0.90
Packed STORE                         1.00
Packed MUL                           1.00
Packed ADD SUB                       1.00
**********************************
EXECUTION PORTS DISPATCH: (cycles)
P0                                   38
P1                                   25
P2                                   57
P3                                   4
P4                                   4
P5                                   25
**********************************
PERFORMANCE PREDICTIONS: (cycles)
L1 prediction :                      57
L2 prediction min :                  NA
L2 prediction avg :                  NA
RAM prediction :                     NA
L1 vect prediction :                 55.25
L2 vect prediction :                 NA
2                                    4.11
14                                   /ARCH/eo/cgsolv2.s, ,
No report for this loop
```

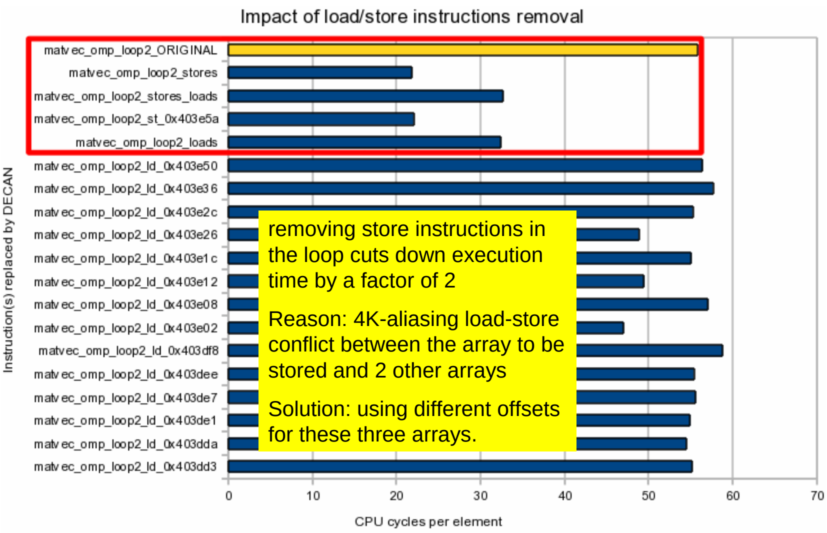
```
movups -8(%r12,%rax), %xmm11 #1
movups -4(%r12,%rax), %xmm1 #19
movups (%r12,%r9), %xmm0 #197.4
movups -4(%r12,%r10), %xmm3 #19
movups -4(%r12,%rdx), %xmm2 #19
movups -4(%r12,%rcx), %xmm5 #19
movups -4(%r12,%r11), %xmm4 #19
movups -4(%r12,%rbx), %xmm7 #20
movups -4(%r12,%rbp), %xmm6 #20
movups -4(%r12,%rsi), %xmm9 #201
movups -4(%r12,%r14), %xmm8 #20
movups -4(%r12,%rdi), %xmm10 #20
movups -4(%r12,%r9), %xmm12 #19
movups 12(%r12,%rax), %xmm14 #1
movups 12(%r12,%rdx), %xmm15 #1
mulps %xmm0, %xmm1 #197.40
movups 12(%r12,%r10), %xmm0 #19
mulps %xmm2, %xmm3 #198.40
movups 8(%r12,%rax), %xmm2 #196
mulps %xmm4, %xmm5 #199.40
mulps %xmm6, %xmm7 #200.40
mulps -8(%r12,%r9), %xmm11 #196.
mulps 8(%r12,%r9), %xmm2 #196.40
mulps %xmm8, %xmm9 #201.40
mulps %xmm15, %xmm0 #198.40
movsd 16(%r12,%r9), %xmm13 #197
movhps 24(%r12,%r9), %xmm13 #19
mulps %xmm13, %xmm14 #197.40
addps %xmm1, %xmm11 #197.21
movups 12(%r12,%rcx), %xmm1 #19
addps %xmm14, %xmm2 #197.21
addps %xmm3, %xmm11 #198.21
addps %xmm0, %xmm2 #198.21
movups 12(%r12,%r11), %xmm0 #19
mulps %xmm0, %xmm1 #199.40
movups 12(%r12,%rbp), %xmm0 #20
addps %xmm5, %xmm11 #199.21
addps %xmm1, %xmm2 #199.21
movups 12(%r12,%rbx), %xmm1 #20
mulps %xmm0, %xmm1 #200.40
movups 12(%r12,%r14), %xmm0 #20
addps %xmm7, %xmm11 #200.21
addps %xmm1, %xmm2 #200.21
movups 12(%r12,%rsi), %xmm1 #201
mulps %xmm0, %xmm1 #201.40
movups 12(%r12,%rdi), %xmm0 #20
addps %xmm9, %xmm11 #201.21
divps %xmm10, %xmm11 #202.15
addps %xmm1, %xmm2 #201.21
divps %xmm0, %xmm2 #202.15
movsd 12(%r12,%r9), %xmm1 #195.
```

Loop highly vectorised

Unrolled by 2 (assembly level)

Lots of movups instructions (unaligned vector load/store)

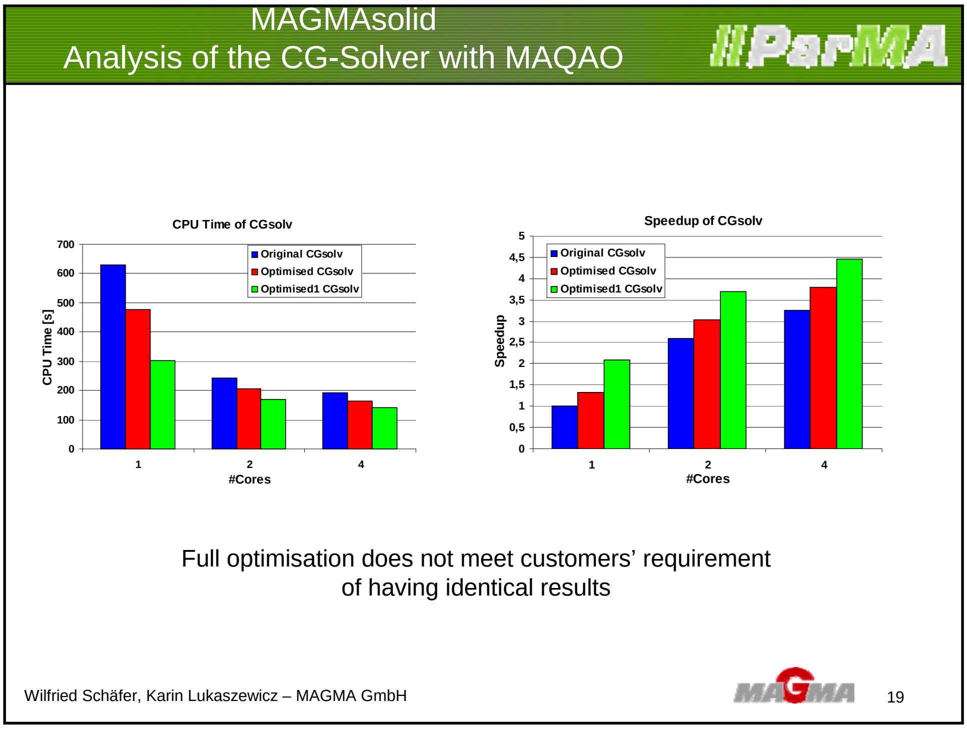
MAGMAsolid
matvec DECAN analysis



removing store instructions in the loop cuts down execution time by a factor of 2

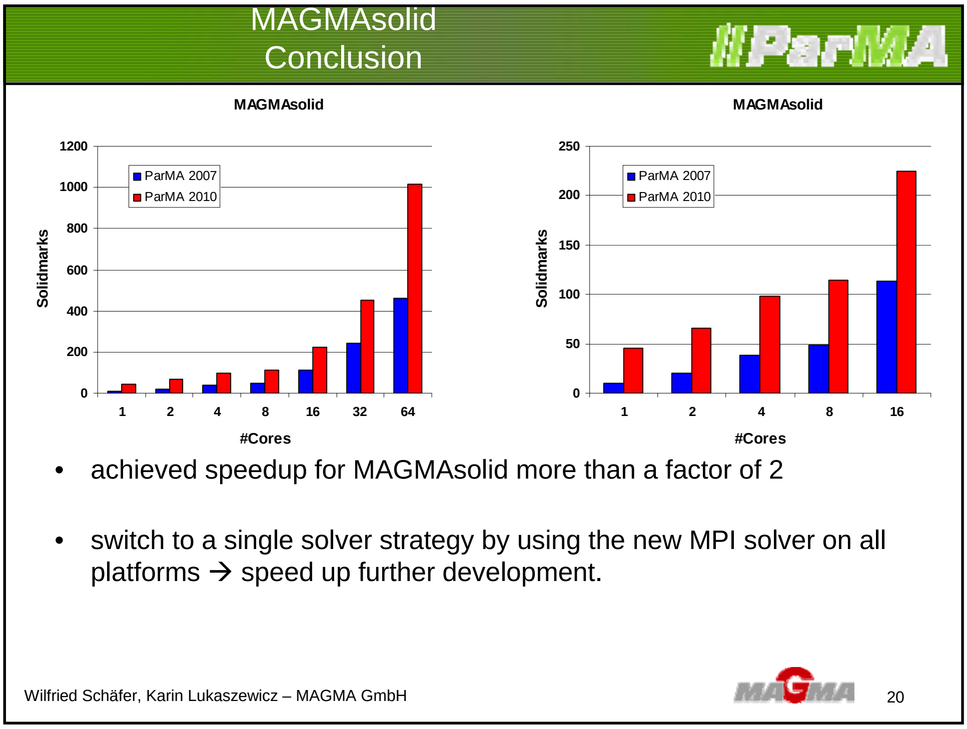
Reason: 4K-aliasing load-store conflict between the array to be stored and 2 other arrays

Solution: using different offsets for these three arrays.



MAGMAsolid
Analysis of the CG-Solver with MAQAO

Full optimisation does not meet customers' requirement of having identical results



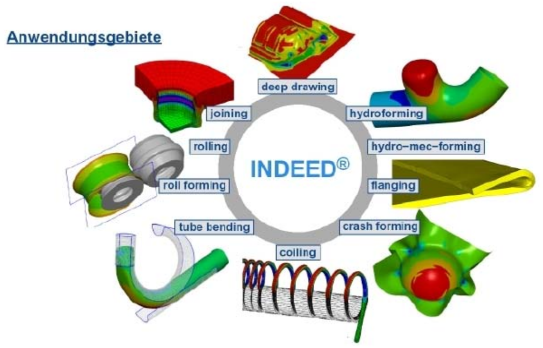
MAGMAsolid
Conclusion

- achieved speedup for MAGMAsolid more than a factor of 2
- switch to a single solver strategy by using the new MPI solver on all platforms → speed up further development.

Content

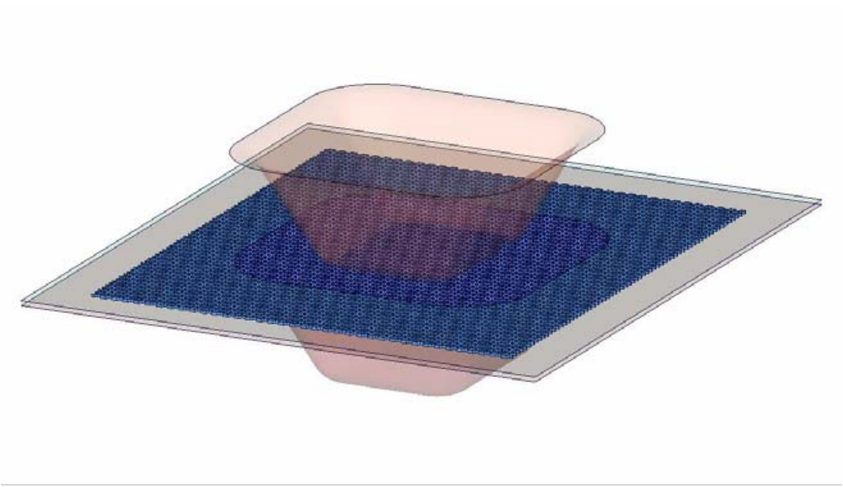
- RECOM – combustion simulation
- MAGMA – casting simulation
- **GNS – metal forming simulation**

INDEED – Applications



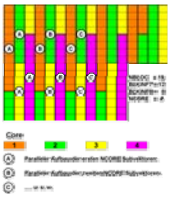
- Implicit Finite Element program for the simulation of metal forming processes.
- Vectorized data structures (Cray, NEC).
- Shared memory parallelism with OpenMP pragmas (since 2000).
- Distributed memory parallelism using MPI standard (started in 2005).

Deep Drawing Simulation

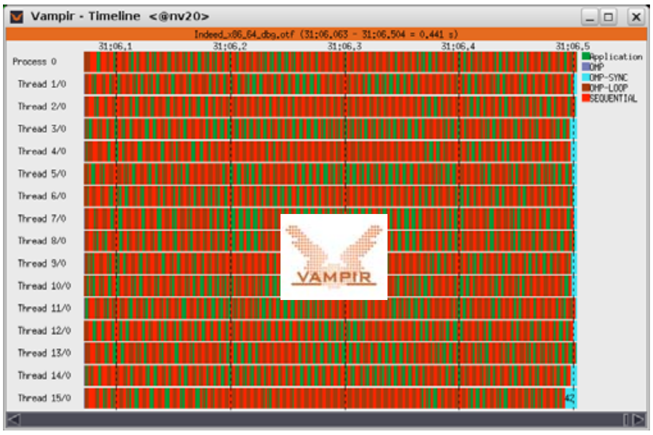


K.Kassem, R.Menzel

OpenMP Version: Improvements (1)


A
B
C
NBLOC = 15
BLKINF7 = 12
BLKINFB = 8
NCORE = 4
Core
1
2
3
4
A Paralleler Aufbau der ersten NCORE Subvektoren
B Paralleler Aufbau der zweiten NCORE Subvektoren
C ... u. s. w.

Analysis with **Scalasca** and **VampirTrace**
showed many sequential amounts in the parallel
Stiffness matrix assembly phase !

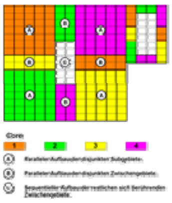


Vampir - Timeline <@nv20>

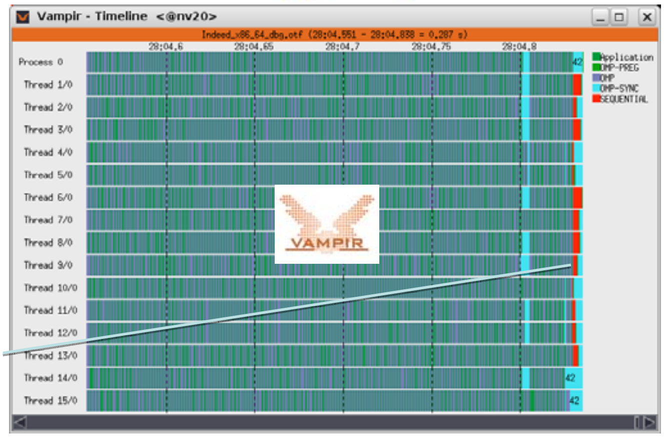
Indeed_x86_64_dbg.otf (31:06.063 - 31:06.504 = 0.441 s)

K.Kassem, R.Menzel

OpenMP Version: Improvements (2)


A
B
C
D
Core
1
2
3
4
A Paralleler Aufbau der disjunkten Subgebiete.
B Paralleler Aufbau der disjunkten Zwischengebiete.
C Sequentieller Aufbau der restlichen sich berührenden Zwischengebiete.

Step 1 (parallel)
CPU0
CPU1
CPUn-1
Step 2 (parallel)
Step 3 (partly parallel)



Vampir - Timeline <@nv20>

Indeed_x86_64_dbg.otf (28:04.551 - 28:04.838 = 0.287 s)

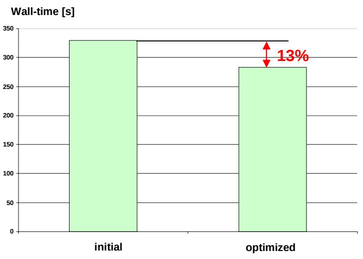
Minimized sequential amounts

OpenMP Version: Improvements (3)

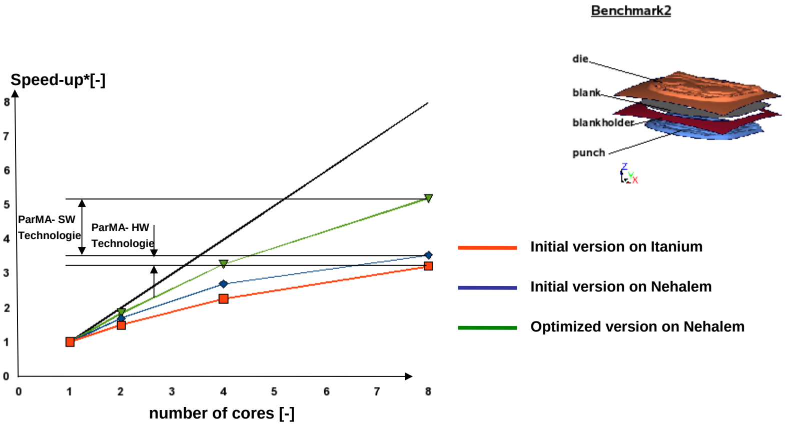
MAQAO analysis: loop within compute intensive routine vectorized insufficiently

Enforce vector alignment

```
      DO 480 J = 1 , 6
cDEC$ VECTOR ALIGNED
         DO 490 I = 1   , NVELE → 64
            VMAT(I,1,J)= -2.D+0*XK32 *
     &                     ( QN(I,1)*NTX0(I,1) * XMAT(I,1,J)
     &                     + QN(I,4)*NTX0(I,4) * XMAT(I,2,J)
     &                     + QN(I,5)*NTX0(I,5) * XMAT(I,3,J)
     &      + (QN(I,4)*NTX0(I,1) + QN(I,1)*NTX0(I,4))* XMAT(I,4,J)
     &      + (QN(I,5)*NTX0(I,1) + QN(I,1)*NTX0(I,5))* XMAT(I,5,J)
     &      + (QN(I,5)*NTX0(I,4) + QN(I,4)*NTX0(I,5))* XMAT(I,6,J) )
!
            VMAT(I,2,J)= -2.D+0*XK32 *
     &                     ( QN(I,7)*NTX0(I,7) * XMAT(I,1,J)
     &                     + QN(I,2)*NTX0(I,2) * XMAT(I,2,J)
     &                     + QN(I,6)*NTX0(I,6) * XMAT(I,3,J)
     &      + (QN(I,7)*NTX0(I,2) + QN(I,2)*NTX0(I,7))* XMAT(I,4,J)
     &      + (QN(I,7)*NTX0(I,6) + QN(I,6)*NTX0(I,7))* XMAT(I,5,J)
     &      + (QN(I,6)*NTX0(I,2) + QN(I,2)*NTX0(I,6))* XMAT(I,6,J) )
      …
```



OpenMP Version: performance gain

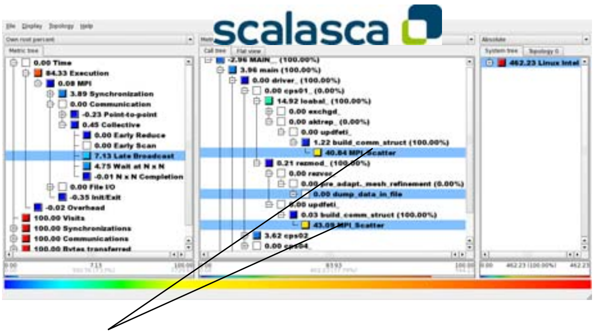


After restructuring the code runs up to 30% faster !!

K.Kassem, R.Menzel

MPI Version: improvements

- ParMA tools showed no severe performance issues despite long computation times → inefficient algorithm.



```
0.00 Time
  84.33 Execution
    0.08 MPI
      3.89 Synchronization
      0.00 Communication
        -0.23 Point-to-point
        0.45 Collective
          0.00 Early Reduce
          0.00 Early Scan
          7.13 Late Broadcast
          4.75 Wait at N x N
          -0.01 N x N Completion
      0.00 File I/O
      -0.35 Init/Exit
  -0.02 Overhead
100.00 Visits
100.00 Synchronizations
100.00 Communications
100.00 Bytes transferred
```

```
-2.96 MAIN__ (100.00%)
  3.96 main (100.00%)
    0.00 driver_ (100.00%)
      0.00 cps01_ (0.00%)
        14.92 loabal_ (100.00%)
          0.00 exchgd_
          0.00 aktrep_ (0.00%)
            0.00 updfeti_
              1.22 build_comm_struct (100.00%)
                40.84 MPI_Scatter
        0.21 rezmod_ (100.00%)
          0.00 rezver_
            0.00 pre_adapt._mesh_refinement (0.00%)
              0.00 dump_data_in_file
          0.00 updfeti_
            0.03 build_comm_struct (100.00%)
              43.05 MPI_Scatter
      3.62 cps02
      0.00 cps04_
```

```
462.23 Linux Intel
```

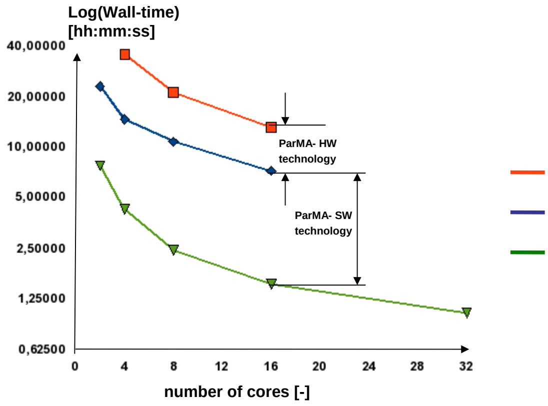
- Integration of **different solver leads to significant stability + performance improvement**

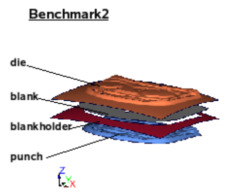
- **Restructuring** of rezoning & load balancing phase leads to **further performance improvement**

Rezoning & load balancing phases: Sequential amounts showed up as bottlenecks

K.Kassem, R.Menzel

MPI Version: performance gain





Due to the modifications the code scales now up to 64 cores.

On 16 cores now only 1.5h are needed instead of 13h !!

K.Kassem, R.Menzel

Thanks for your attention

Special thanks to all members of the ParMA project for the excellent work done in the past 3 years…

…and coming soon: our follow-up project
H4H (Hybrid for Heterogeneous)

More information

- **ParMA website: http://www.parma-itea2.org**
  - ParCo 2009, Minisymposium “Parallel Programming Tools for Multi-core Architectures”
  - Several papers presenting results with tools and applications
  - http://www.parma-itea2.org/index.php?option=com_content&task=view&id=67&Itemid=35
- **UNITE website: http://apps.fz-juelich.de/unite/**