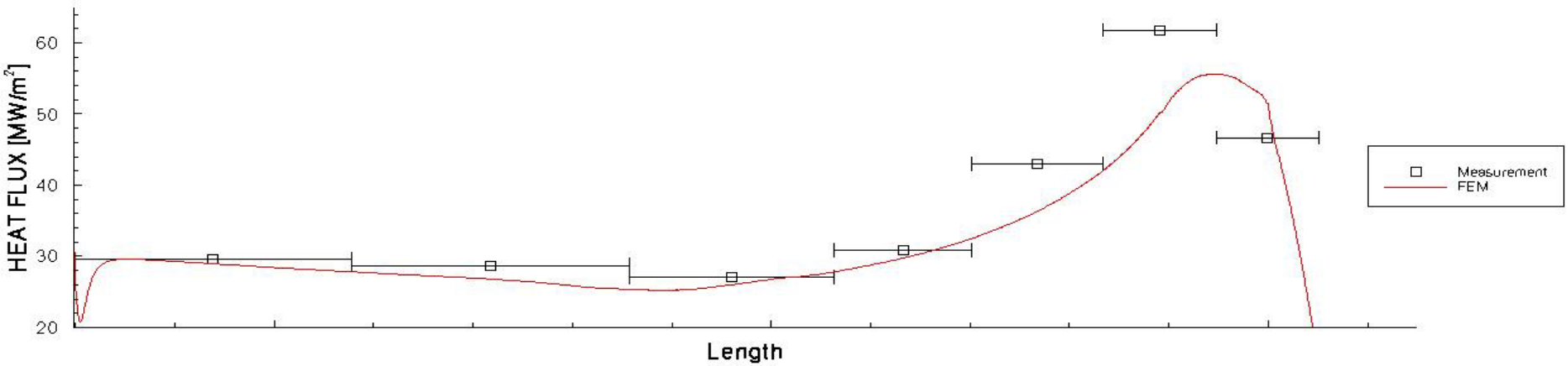
The Thermoflow60 Finite-Element Program

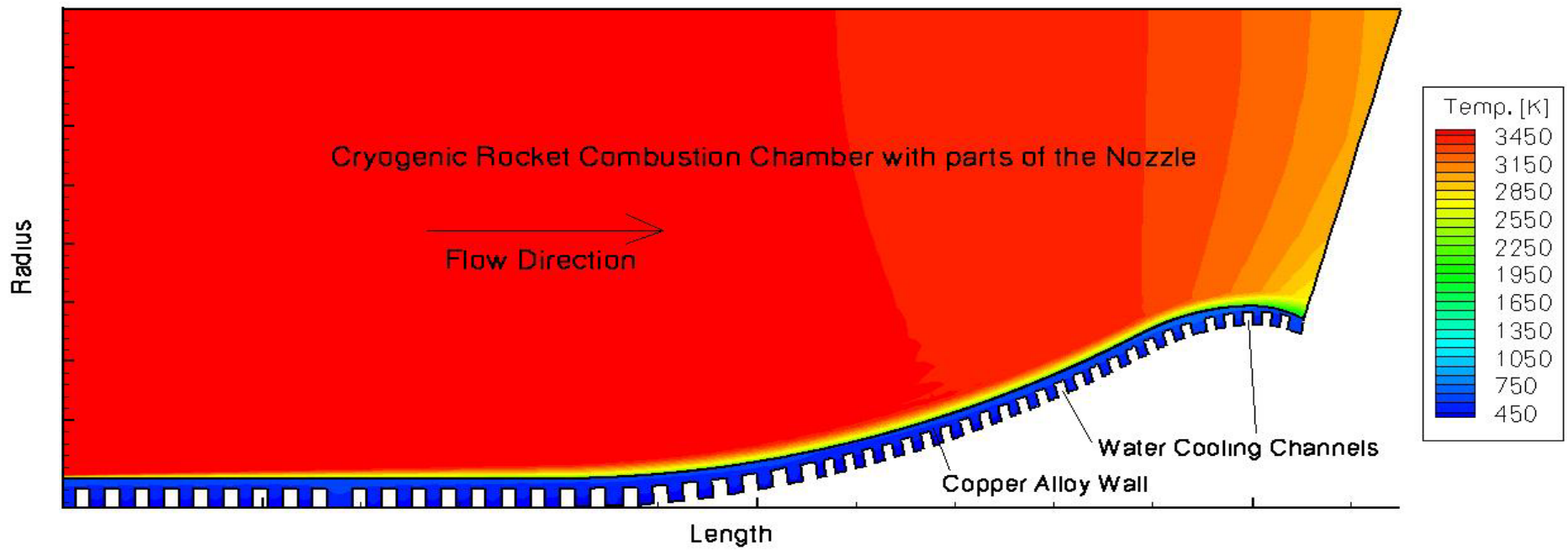
Ulrich Wepler, German Aerospace Center (DLR)
Dieter an Mey, Center for Computing and Communication, RWTH
Alexander Spiegel, Center for Computing and Communication, RWTH
Thomas Haarmann, DaimlerChrysler
Wolfgang Koschel, Jet Propulsion Laboratory, RWTH

Heat Flow Simulation with Finite Elements - ThermoFlow60

- simulation of the heat flow in a rocket combustion chamber
- 2D sufficient because of rotational symmetrie
- Finite Element method
- home-grown code
- 14 years of development
- has been vectorized before
- 29000 lines of Fortran
- ~ 200 OpenMP directives
- 69 parallel loops
- 1 main parallel region (orphaning)
- 200,000 cells
- 230 MB memory footprint
- 2 weeks serial runtime

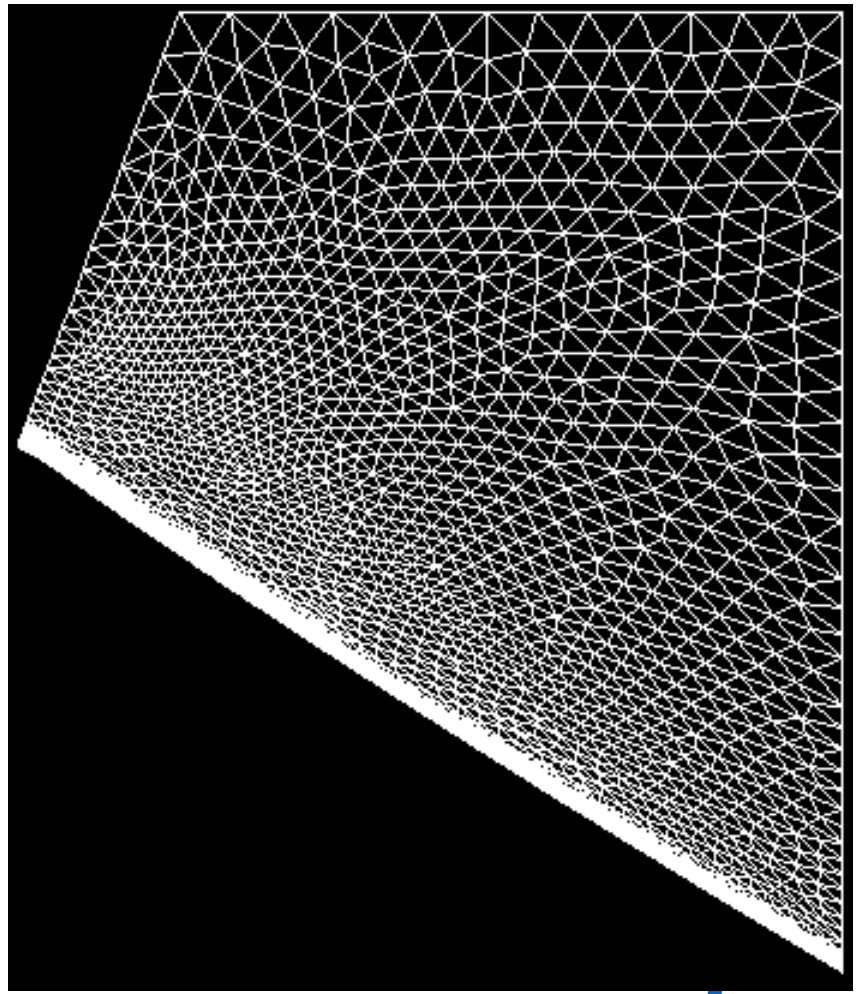
Simulation of the Heat Flow in a Rocket Combustion Chamber





The Grid

- very fine resolution at the boundary to the wall.
- no adaptation necessary
  =>
  loop limits remain constant
- coupled CFD- and structural analysis in preparation



First Approach to OpenMP

```
c$omp parallel do private(k1,k2,k3,dtel,q1,q2,q3,dtdrye,viture,ynorme,retur
c$omp&            rhoe,tuke,epse,XMATL,PRODE1,prode2,dampqe,cw1e,q11,q12,
c$omp&            q13,w11,w12,w13,VILAML,VITURL,RLAMDA,RMUE2,DUDXNS,
C$omp&            DUDYNS,DVDXNS,DVDYNS,DTDX,DTDY,dq2dx,dq2dy,CPL,UUEL,
C$omp&            VVEL,VISK,VISE,q21,q22,q23,w21,w22,w23)
          DO I=1,NELM
             K1     = IELM(I,1)
             K2     = IELM(I,2)
             K3     = IELM(I,3)
             DTEL   = .5d+00*DRI*(DTKN(K1)+DTKN(K2)+DTKN(K3))
             Q1     = U(K1)*DNDX(I,1)+V(K1)*DNDY(I,1)
             Q2     = U(K2)*DNDX(I,2)+V(K2)*DNDY(I,2)
             Q3     = U(K3)*DNDX(I,3)+V(K3)*DNDY(I,3)
             DTDRYE = DRI/YEL(I)
              ---  129 lines omitted ---
             q21 = tukl(i) * cq1 * cmy*dampqe*prode1*rhol(i)/epsl(i)
             q22 = tukl(i) * cq1 * prode2
             q23 = tukl(i) * cq1 * ( 1.0d+00+sark*xmatl )*epsl(i)/rhol(i)
             w21 = epsl(i) * cw1e * cmy*prode1*rhol(i)/epsl(i)
             w22 = epsl(i) * cw1e * cw3*1.5d+00*prode2
             w23 = epsl(i) * cw2*epsl(i)/rhol(i)

             qtukl(i) = q21 + q22 - q23
             qepsl(i) = w21 + w22 - w23
          END DO
c$omp end parallel do
```

loop over all elements

All these arrays reside in (shared) COMMON blocks

Here Orphaning simplifies the Code ...

code out of parallelized loops has to be put in single regions or has to be executed redundantly

```
c$omp do
      DO I=1,NELM
         K1      = IELM(I,1)
         K2      = IELM(I,2)
         K3      = IELM(I,3)
         DTEL    = .5d+00*DRI*(DTKN(K1)+DTKN(K2)+DTKN(K3))
         Q1      = U(K1)*DNDX(I,1)+V(K1)*DNDY(I,1)
         Q2      = U(K2)*DNDX(I,2)+V(K2)*DNDY(I,2)
         Q3      = U(K3)*DNDX(I,3)+V(K3)*DNDY(I,3)
         DTDRYE  = DRI/YEL(I)
          ---  129 lines omitted ---
         q21 = tukl(i) * cq1 * cmy*dampqe*prode1*r    l(i)/epsl(i)
         q22 = tukl(i) * cq1 * prode2
         q23 = tukl(i) * cq1 * ( 1.0d+00+sark*xmatl )*ep     (i)/r    l(i)
         w21 = epsl(i) * cw1e * cmy*prode1*rhol(i)/epsl(i)
         w22 = epsl(i) * cw1e * cw3*1.5d+00*prode2
         w23 = epsl(i) * cw2*epsl(i)/rhol(i)

         qtukl(i) = q21 + q22 - q23
         qepsl(i) = w21 + w22 - w23
      END DO
c$omp end do
```

all the local variables are private by default

All these arrays in COMMON blocks remain shared

Frequently used Loop Constructs

- Loop type 1, loop over (~100,000) FE nodes

```
!$omp do
do i = 1, npoin
   ...
end do
!$omp end do
```

- Loop type 2, loop over (~200,000) FE cells

```
!$omp do
do i = 1, nelm
   ...
end do
!$omp end do
```

- Loop (nest) type 3, loop over nodes and neighbours

```
!$omp do
do i = 1, npoin
   do j = 1, nknot(i)  ! varies between 3 and 6
      ...
   end do
end do
!$omp end do
```

Eliminating unnecessary Barriers

! Barriers between loops of the same type
! can in many cases be eliminated:

```
!$omp do
do i = 1, npoin
   ...
end do
!$omp end do nowait
```

```
!$omp do
do i = 1, npoin
   ...
end do
!$omp end do
```

Avoiding the Overhead of Worksharing Constructs

- Loop type 1, loop over (~100,000) FE nodes

```
!$omp do                                  do i = ilo_poin, ihi_poin
do i = 1, npoin                  --->        ...
   ...                                    end do
end do                                    !$omp barrier
!$omp end do
```

- Loop type 2, loop over (~200,000) FE cells

```
!$omp do                                  do i = ilo_elm, ihi_elm
do i = 1, nelm                   --->        ...
   ...                                    end do
end do                                    !$omp barrier
!$omp end do
```

- Loop (nest) type 3, loop over nodes and neighbours

```
!$omp do                                  do i = ilo_knot, ihi_knot
do i = 1, npoin                              do j = 1, nknot(i)
   do j = 1, nknot(i)            --->           ...
      ...                                    end do
   end do                                 end do
end do                                    !$omp barrier
!$omp end do
```

Precalculating the Loop Limits (1 of 2)

```
!  Loop type 1, loop over (~100,000) FE nodes
      integer ilo_poin,ihi_poin,ilo_elm,ihi_elm,ilo_knot,ihi_knot
      common /omp_com/ilo_poin,ihi_poin,ilo_elm,ihi_elm,ilo_knot,..
!$omp threadprivate(/omp_com/)

      nrem_poin   = mod ( npoin, nthreads )   ! remaining nodes
      nchunk_poin = ( npoin - nrem_poin ) / nthreads   ! chunk size

!$omp parallel private(myid)
      myid = omp_get_thread_num()
      if ( myid < nrem_poin ) then
              ilo_poin = 1 + myid * ( nchunk_poin + 1 )
              ihi_poin = ilo_poin + nchunk_poin
      else
              ilo_poin = 1 + myid * nchunk_poin + nrem_poin
              ihi_poin = ilo_poin + nchunk_poin  - 1
      end if
!$omp end parallel

!  Loop type 2, loop over (~200,000) FE cells
     --- similar to loop type 1 ---
```

Even work distribution

Precalculating the Loop Limits (2 of 2)

! **Loop (nest) type 3, loop over nodes and neighbours**

```
      itotal = 0
      do i = 1, npoin
         itotal = itotal + nknot(i)
      end do
      nchunk_knot = itotal / nthreads

      itotal = 0
      ithread = 0
      ilo_temp(0) = 1
      do i = 1, npoin
         itotal = itotal + nknot(i)
         if ( itotal .ge. (ithread+1)*nchunk_knot ) then
            ihi_temp(ithread) = i
            ithread = ithread + 1
            if ( ithread .ge. nthreads ) exit
            ilo_temp(ithread) = i + 1
         end if
      end do
      ihi_temp(nthreads-1) = npoin
!$omp parallel private(myid)
      myid = omp_get_thread_num()
      ilo_knot = ilo_temp(myid)
      ihi_knot = ihi_temp(myid)
!$omp end parallel
```

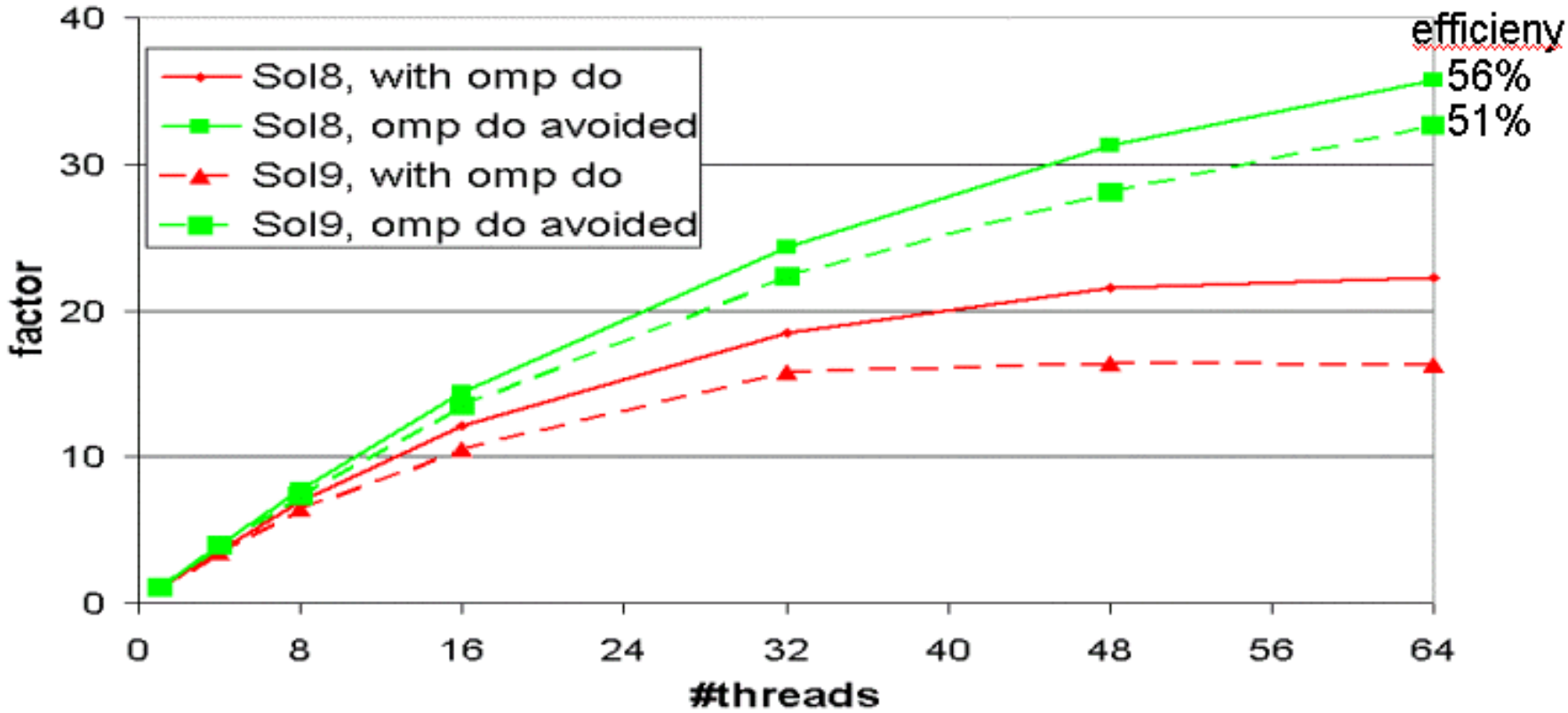
Loop Nest with Precalculated Optimal Schedule

```
   do i = ilo_knot, ihi_knot
      do j = 1,nknot(i)
          ii   = iknot(i,j)     ! Element number
          kk   = iknel(i,j)     ! local node number (1-3)

      ---  28 lines omitted ---

      end do
   end do
c$omp barrier
```

Timings on Sun Fire 15K



OMPlab

- Scalability on Sun and IBM
- Analysis (Sun Analyzer, IBM xprofiler, Paraver ...)
- Memory Placement
- Tuning