The Thermoflow60 Finite-Element Program

Ulrich Wepler [1], Dieter an Mey[2], Thomas Haarmann[3], Wolfgang Koschel[4]

1) German Aerospace Center (DLR) [1]
2) Center for Computing and Communication, Aachen University (RWTH) [2]
3) DaimlerChrysler, Stuttgart, Germany
4) Jet Propulsion Laboratory, Aachen University (RWTH) [3]

1. Introduction

Good vectorizable code employs clean data structures and long loops with many operations and thus is also well suited for loop-level shared-memory parallelization with OpenMP. Nevertheless the scalability of such an approach is in many cases limited, because of the overhead of many parallel regions and worksharing constructs involved, unless measures are taken to overcome these limitations. In order to improve the scalability of Finite-Element (FE) program Thermoflow60, used for the simulation of the heat flux in a rocket combustion chamber, we employed one parallel region which contains the bulk of the computation. Precalculation of the limits of those loops, which are frequently executed, furthermore reduces the overhead of the worksharing constructs and improves the scalability ([5]).

2. Parallelization of the Thermoflow60 Finite-Element Program using OpenMP

Over the last 14 years a Finite Element CFD solver has been developed at the Jet Propulsion Laboratory at the Aachen University. In the early days this application was used to simulate all kind of internal and external flows. In its further development the simulation of wall heat fluxes became a new goal which was finally performed by solving the "fluid" domain and the "structure" domain in a coupled manner. Today the code is used to simulate heat transfer problems in rocket combustion chambers (fig. 1) [4]. Because this is a true rotational symmetric problem a 2D simulation meets the necessary accuracy. In order to determine the heat fluxes correctly, certain physical effects in the boundary layer require a very fine grid in the near wall region. This leads to grids with several 100.000 cells. These simulations can only be performed in a justifiable timeframe by using parallelization.

Because of the availability of Fujitsu-Siemens vector systems at the Aachen University throughout the nineties the program has been well adapted to the specific vector computer architecture. With the replacement of the last vector machine by a Sun Fire SMP cluster at Aachen it was necessary to parallelize the code. Fortunately OpenMP as the new de-facto standard for shared-memory parallelization is now available and mature enough to justify the investment of man power needed for parallelization and loop-level parallelization with OpenMP also is a natural replacement for vectorization, as both profit from clean data structures and long "fat" loops.
The current OpenMP version of Thermoflow60 consists of 29000 lines of Fortran with about 200 OpenMP directives, containing 69 parallel loops overall.

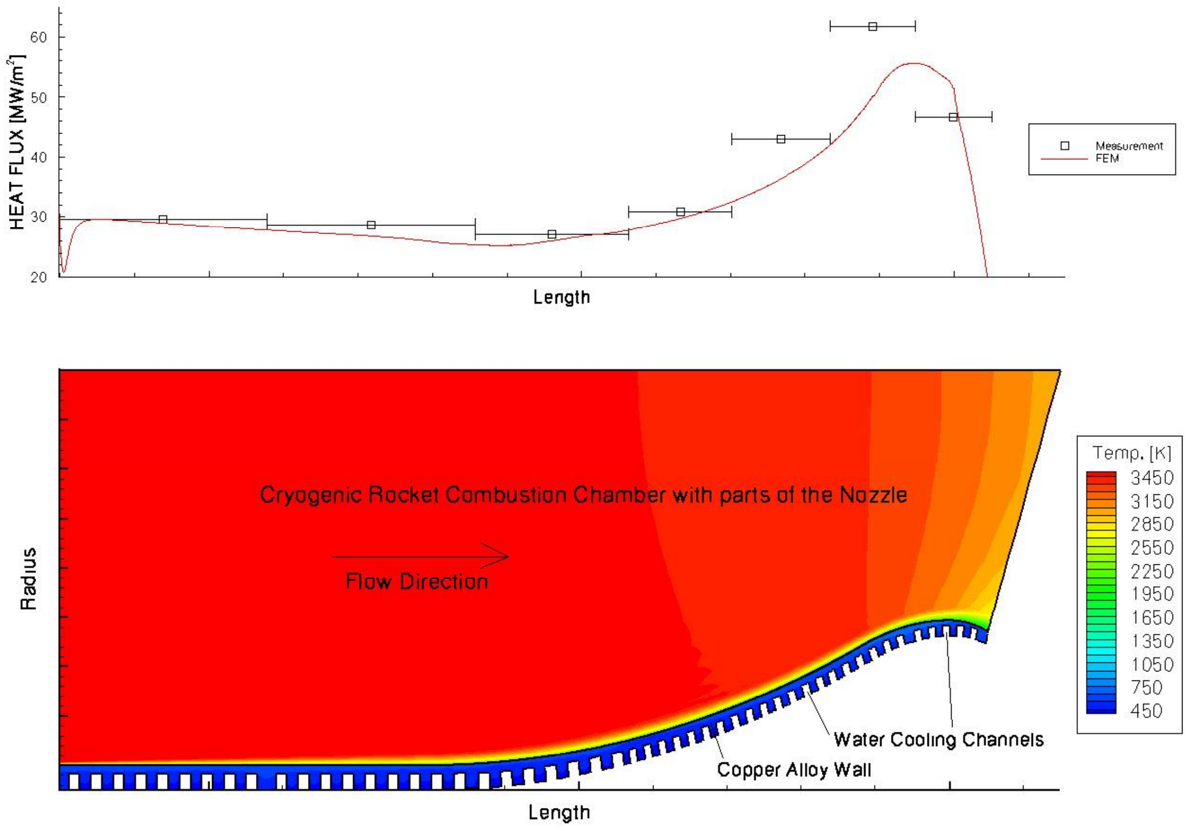


Fig. 1: Heat flow in a rocket combustion chamber

Expanding the parallel regions

The first OpenMP approach was quite cumbersome. Introducing parallel regions and worksharing constructs around the many loops not only was a lot of work but also very error-prone. Like other typical CFD codes written in (extended) Fortran77, all large global arrays had been put into common blocks.
Many subroutines contain heavy calculations in long loops reading and modifying these global arrays with the help of many locally declared scalar temporaries, which in a loop-level parallelization have to be declared private.

In the following versions the parallel regions were extended. As soon as the parallel regions are extracted out of the subroutines containing the parallel loops (orphaning) the default rules for variable scoping change: Whereas in the static extent of a parallel region all these temporary scalar variables are shared by default and need to be privatized explicitly, they are private by default in the dynamic extend. Suddenly all the long lists in private clauses of the do directives vanish.
This strategy was followed until finally the whole iteration loop containing almost all of the calculation fitted into one single parallel.

Avoiding the worksharing `omp do` construct

Revisiting the compute intensive parts, it turned out that most loops fall into two categories: loops over the number of nodes of the underlying 2D Finite Element grid on one hand (loop type 1) and loops over the number of cells on the other hand (loop type 2).

If successive loops fall into the same category, there is a good chance that a barrier in between can be avoided with a `nowait` clause, if the loop scheduling is static, thus increasing the scalability.

Because the problematic areas of the simulation, the boundary layers, are well-known in advance, the grid has a very fine resolution in these areas from the very beginning and adaptation during the simulation process is not necessary.

As a consequence the loop ranges do not change over time. This allows a precalculation of the loop chunks for all threads for these two loop categories and the elimination of all corresponding worksharing `do` directives. Only the `end do` directives without a `nowait` clause have to be replaced by a `barrier` directive in order to insure the necessary synchronisations.

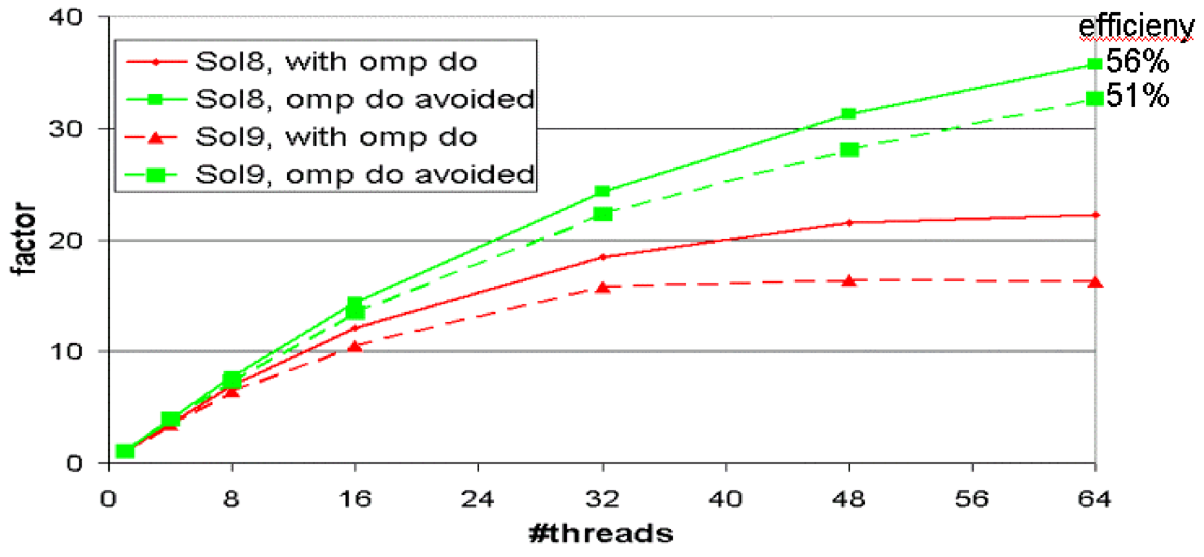
Some time consuming loop nests have a special structure which might be further exploited (loop type 3). Whereas the outer loop runs over the number of nodes, the inner loop runs over the number of cells this specific node belongs to, which may vary depending on its position in the grid. Particularly nodes on the boundaries typically belong to fewer cells than inner nodes (in many cases 4 compared to 6). This might cause an imbalance of the parallelized outer loop. As the grid does not change over time, an optimal schedule can be precalculated once. But it turned out, that in this case, the difference between an "optimal" schedule and a static work distribution does not pay off.

Timing measurements

Timing measurements have been carried out

- on a Sun Fire 15K with 72 UltraSPARC-III/Cu processors with a 900 MHz clock cycle using Sun's ONE Studio 7 Fortran95 compiler under Solaris 8.

The pre- and postprocessing phases have been neglected and only a few iterations of the compute intense part have been measured, because they clearly dominate real production runs which take many hours.
The measurements show that it is possible to write a scalable OpenMP program with "only" loop level parallelism, meaning that the worksharing is done around inner loops or loop nests in the leaves of the program tree, if the parallel region can be extended.
A speed-up factor of almost 40 on 64 CPUs on the Sun Fire 15K or equivalently an efficiency of up to 56 percent are encouraging results (fig. 2)



.

Fig. 2: Speedup on the Sun Fire 15K system

3. Objectives of the Participation at OMPlab

During the OMPlab we want to further analyze and improve the performance of the Thermoflow60 code. We want to compare the efficiency on different platforms and investigate the effect of different memory placement strategies on NUMA architectures.

Literature

1) German Aerospace Center http://www.dlr.de
2) Center for Computing and Communication, Aachen University of Technology http://www.rz.rwth-aachen.de
3) Jet Propulsion Laboratory, Aachen University of Technology http://www.bst.rwth-aachen.de
4) Haarmann T.M., Koschel W.W.: "Numerical Simulation of Heat Loads in a Cryogenic H_2O_2 Rocket Combustion Chamber", PAMM Proceedings in Applied Mathematics and Mechanics, 2(1): 360-361,2002
5) Dieter an Mey [1], Thomas Haarmann[3], Wolfgang Koschel[2]: Pushing Loop-Level Parallelization to the Limit, 4th European Workshop on OpenMP, Rome, 2002