NUCLEAR SAFETY INSTITUTE OF RUSSIAN
ACADEMY OF SCIENCES
IBRAE RAN

MELCOR 2.1

Code performance improvement

Dr. Nastasia Mosunova
Dr. Andrey Gorobets
Irina Drobyshevskaya

EMUG-2011
Bologna, Italy
April, 12, 2011

Outline

- Linear solver replacement
- Critical flow model modernization
- Code refactoring
- Parallelization of CVH package with OpenMP
- Performance testing

Goal of the work

- To find the ways of code performance improvement
- To apply the proposed modifications
- To test and analyze obtained calculation results

Linear solver replacement

- Changes in matrix representation
- PARDISO solver implementation
- Changes in user input and output
- Testing

Matrix representation replacement

- The testing has shown that about 50% of CVH package CPU time is consumed in a call to a sparse linear solver (DSDBCG - is the bi-conjugate gradient method)

- Original matrix representation
  - The matrix is formed in tHydr_CVHMOM each time it is called
  - The matrix is in dense format of $O(N^2)$ cost in terms of memory
  - The matrix is converted later on in $O(N^2)$ operations to some internal sparse format

- Matrix representation replacement
  - The matrix is formed in O(N) sparse Pre-CSR format (which allows fast addition of elements)
  - Then matrix is converted to CSR format at O(N) cost

PARDISO solver

It is a direct LU-based parallel solver from the MKL library supplied with the Intel compiler.

- It consists of
  - **Analysis stage**
    Consumes most of time (around 2/3) and it is not parallel. Mainly it is reordering algorithm for a sparse matrix. It is only needed when the matrix structure changes.
  - **Factorization stage**
    Consumes around 1/3 of total solver time. It is called every time. It is parallel but not for that small sizes. See performance details further.
  - **Solution stage**
    Consumes not a lot comparing with stages 1 and 2.
  - **Deallocation of data.**

Changes in user input and output (1)

- The solver type can be changed
  - With the command line argument ST=<1 or 2>:
    - *1 – DSDBCG solver (default)*
    - *2 – PARDISO solver*
- Through sensitivity coefficient SC4420

4420 – A sparse linear solver type and matrix checker for direct parallel solver

The coefficient is used to specify the sparse linear solver type. Currently two options are available: bi-conjugate gradient method or direct parallel solver. In case the direct parallel solver is chosen additional solver parameters could be set through the other sensitivity coefficient elements.

(1) - Flag that defines the solver type:
1 – bi-conjugate gradient method;
2 – direct parallel solver
(default = 1.0, units = dimensionless)

(2) - Matrix checker.
0 – direct parallel solver does not check the sparse matrix representation;
1 – direct parallel solver checks whether column indices are sorted in increasing order within each row.
(default = 0.0, units = dimensionless)

Changes in user input and output (2)

- Examples
  - from the command line:

    Melcor.exe *.inp ST=2

  - through sensitivity coefficient

```
CVH_SC 2 !   N Name     Value       Index
             1   4420     2.0         1
             2   4420     1.0         2
```

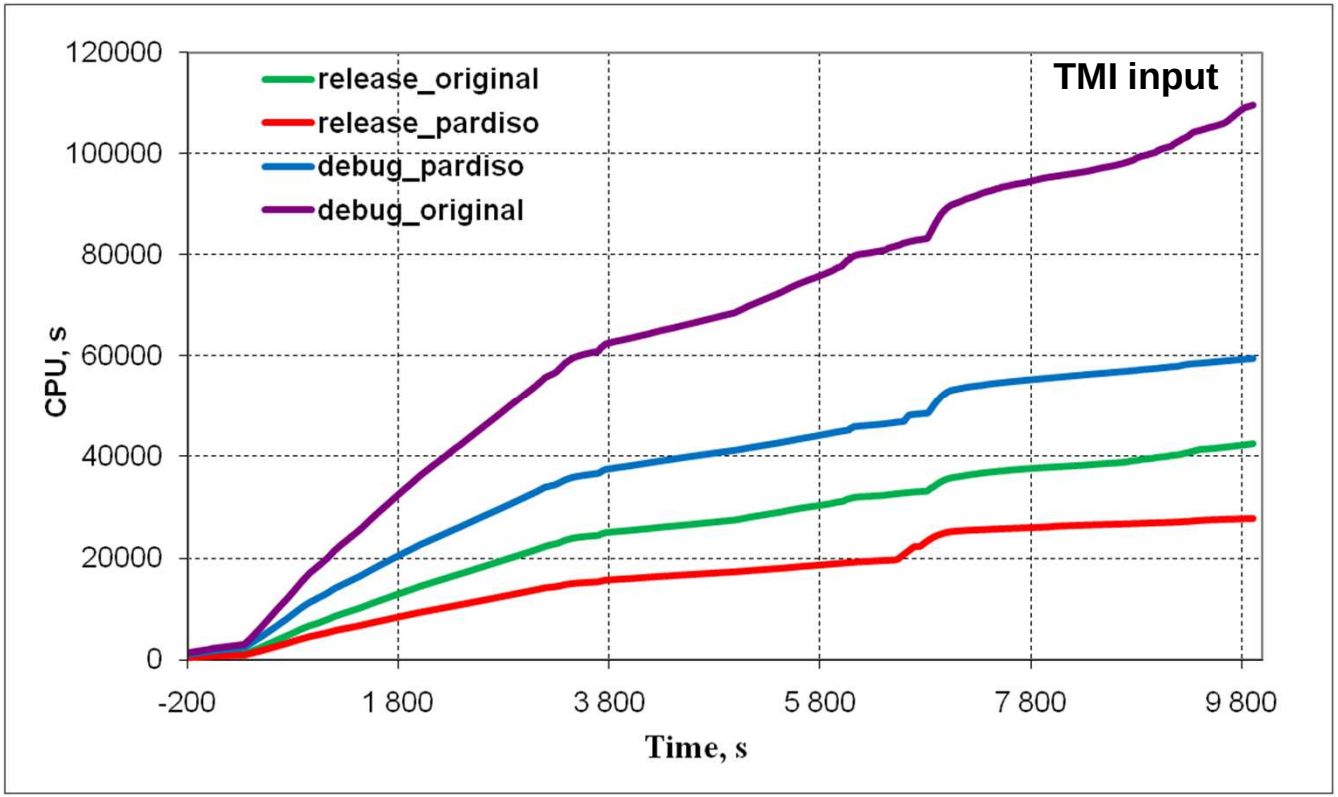
- Changes in the output

CVH PACKAGE :: PARDISO solver (direct parallel solver) is used in cvhmom as defined by the user

CVH PACKAGE :: DSDBCG solver (bi-conjugate gradient method) is used in cvhmom as defined by the user

Testing and future development

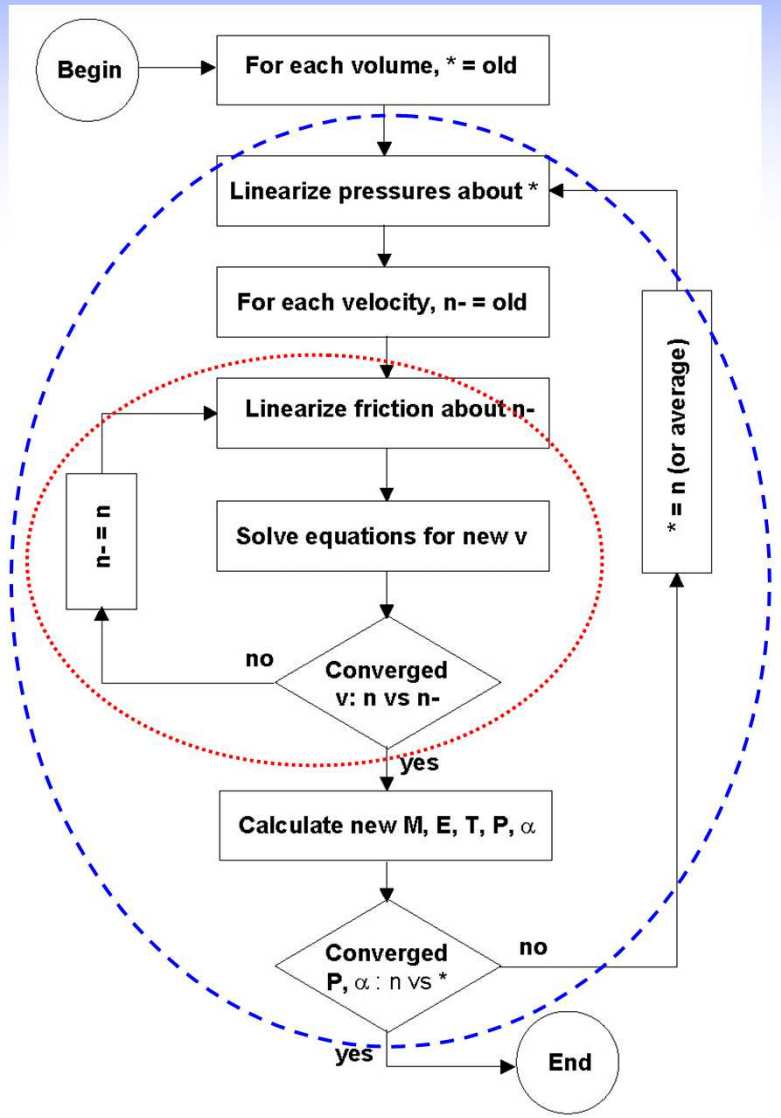
- Wide solver testing
- Other solvers are under investigation (solver based on stabilized bi-conjugate gradient method (BICG-STAB), UMFPAC solver, etc.)



Critical flow model modernization

- Modernized procedure for critical velocity calculation
- Changes in user input
- Testing

Modernized algorithm (1)



Modernized algorithm (2)

- Calculation of “critical” pressure difference value

$$\Delta P_{critical,j} = \xi_j \frac{\rho_j c_{s,j}^2}{2}$$

- Comparison of “critical” pressure difference value with actual pressure difference obtained as difference of linearly projected new pressures. If

$$\Delta P_j^{\tilde{n}} > \Delta P_{critical,j}$$

correction of local friction (form) loss coefficient to limit flow velocity by critical one

$$\xi_j^{corrected} = \xi_j \frac{\Delta P^{\tilde{n}}_j}{\Delta P_{critical.j}}$$

- Using of new (corrected) local friction (form) loss coefficient in global solution with no sub-iteration

User input modifications

- Additional (temporary) sensitivity coefficient SC4450 is introduced to provide a possibility to chose between using old procedure for critical flow calculation and modernized one.

```
CVH_SC   1  ! N   SCnumber   Value      Index
            1      4450        1.0        1  ! 1 – moder.; 0 - old
```

- New field on FL_USL record to provide a possibility to choose either to verify if choking exists in given flow path or to ignore it.

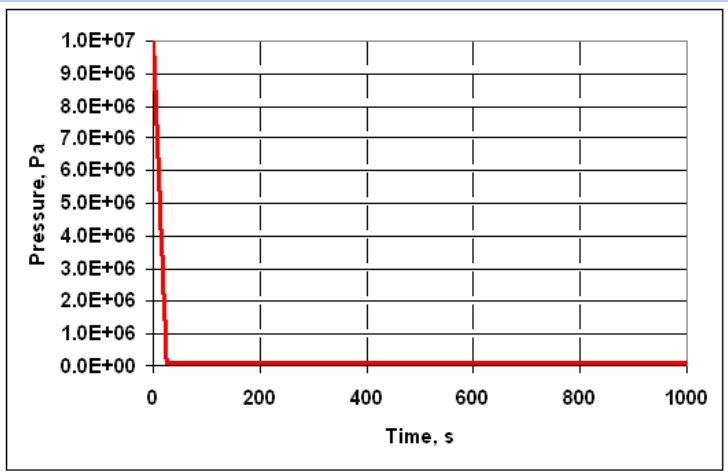
```
                                  ! New field
FL_USL 1.  1.  1.  1.  CHOKING  ! CHOKING explicitly set
FL_USL 1.  1.  1.  1.  IGNORECHOKING  ! CHOKING is off
```

Modernized critical flow model: Testing (1)

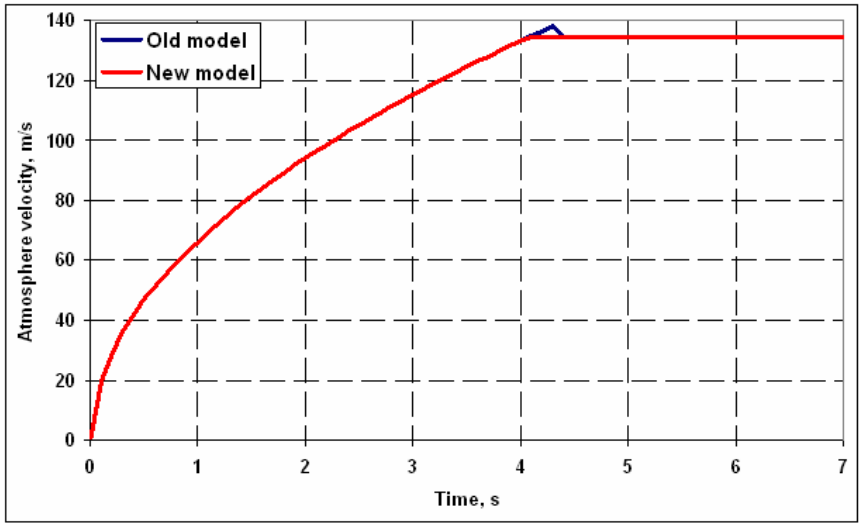
Time-independent CV "From"
100% N_2
P = 10^7 Pa
T = 300 K

FL

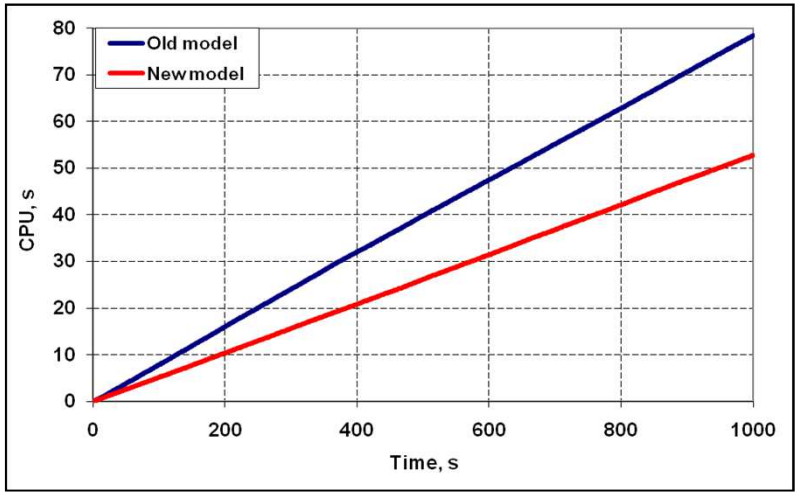
Properties-specified CV "To"
100% N_2
P = CF
T = 300 K



Pressure in "To" CV



Atmosphere velocity through the break for the first 7 s

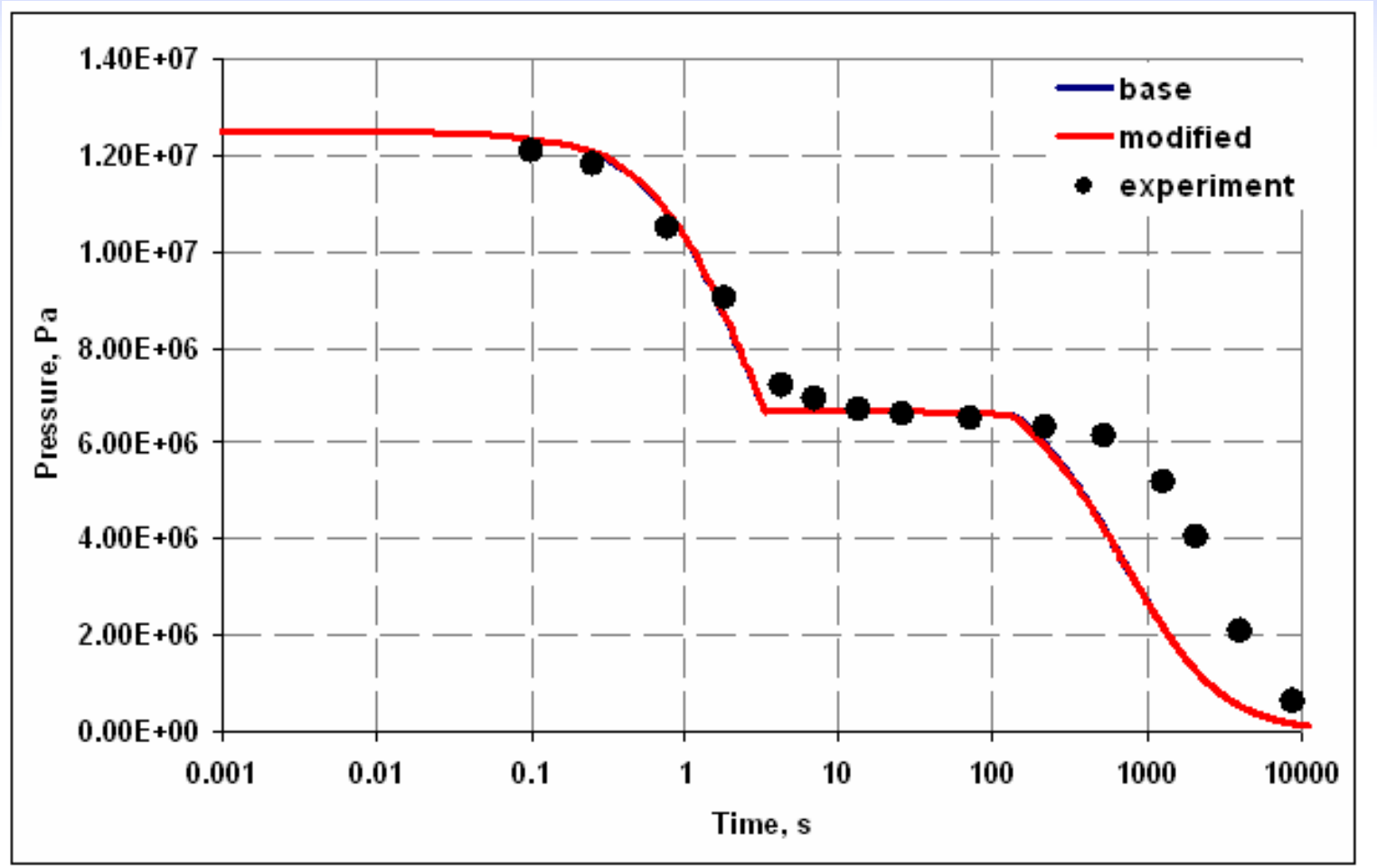


Total computational time

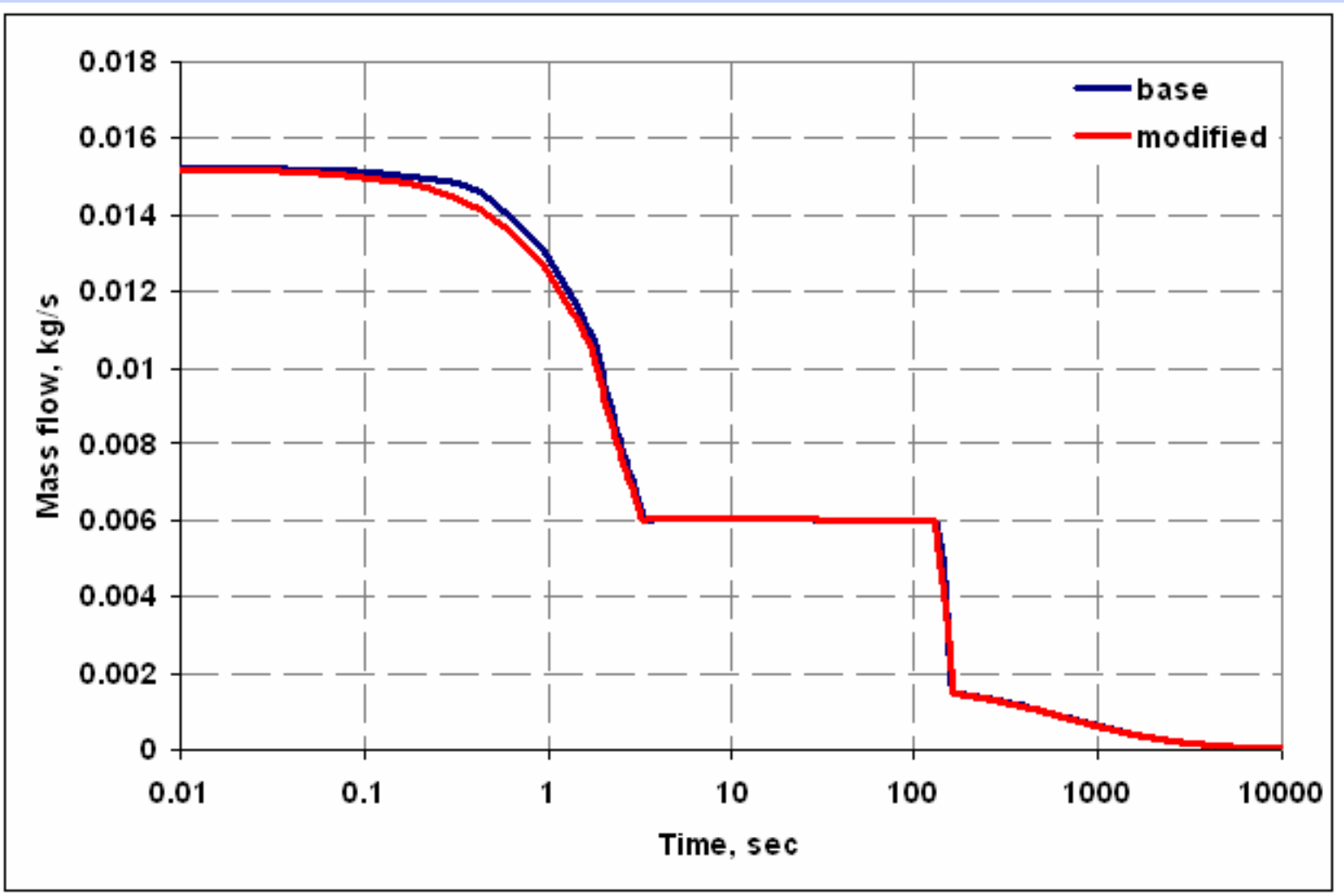
Modernized critical flow model: Testing (2)

Test	Problem time, s	CPU time “base”, s	CPU time “modified”, s
Small TF leaks			
0.5 mm	11000.	56.078	44.594
2 mm	1000.	5.203	5.375
5 mm	1000.	5.828	5.672
10 mm	1000.	5.938	5.625
25 mm	1000.	6.48	5.78
Large TF leaks			
5 mm	3000.	3.375	2.766
10 mm	3000.	3.0	3.0
15 mm	3000.	3.672	3.203
25 mm	3000.	4.0	3.656

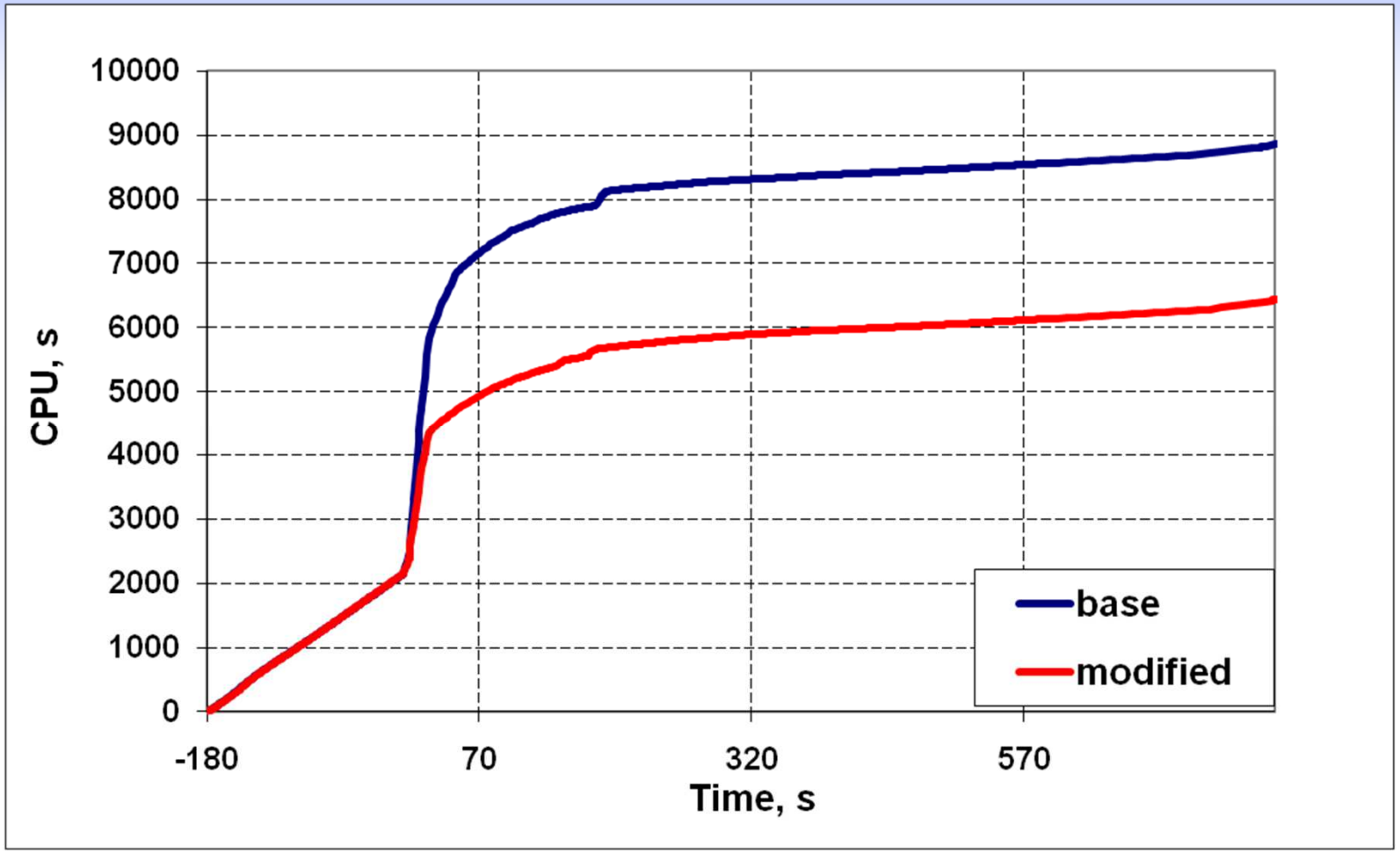
MEI experiment: pressure for the “STF 0.5 mm leak” test



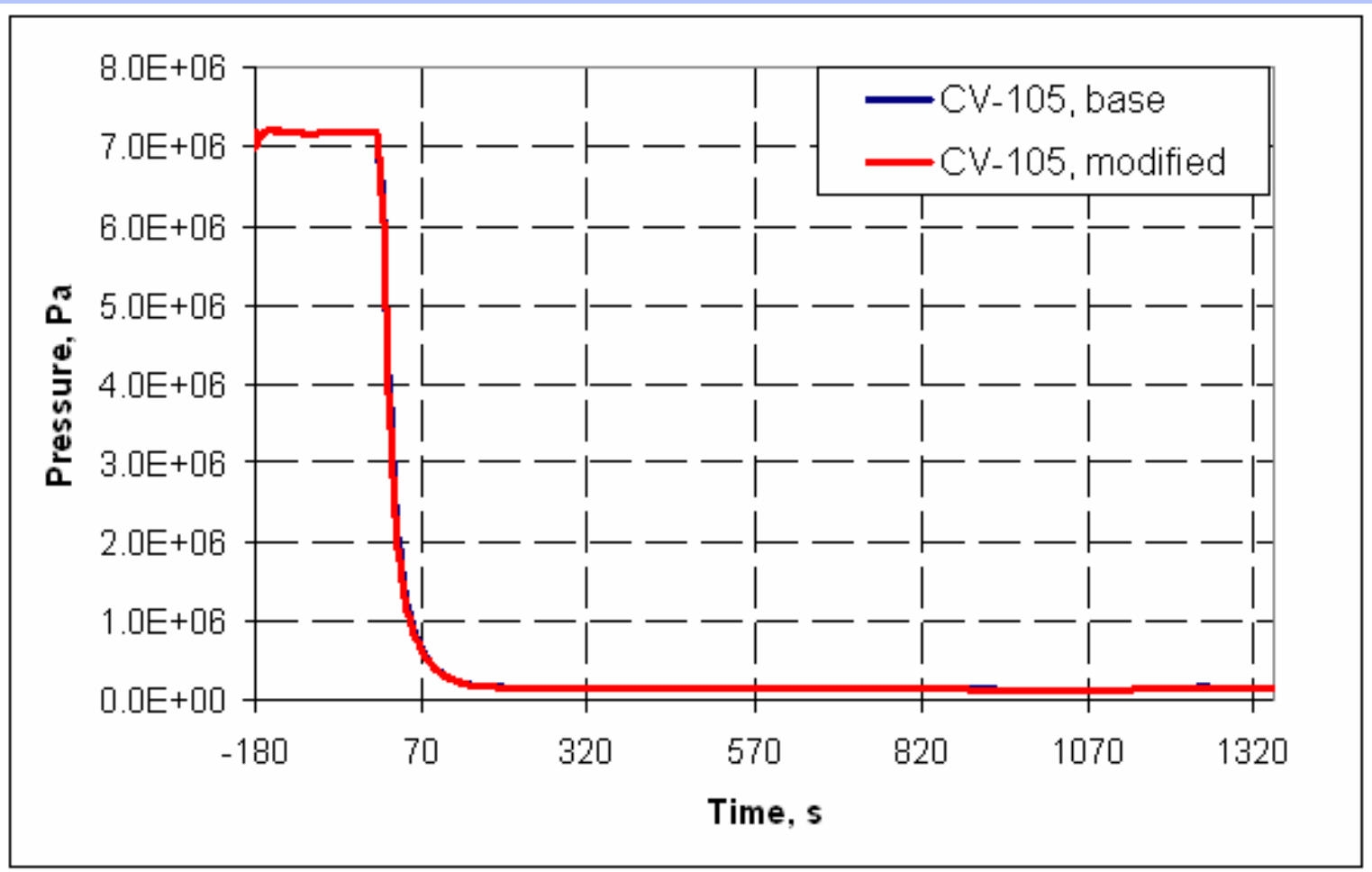
MEI experiment: mass flow rates for the “STF 0.5 mm leak” test



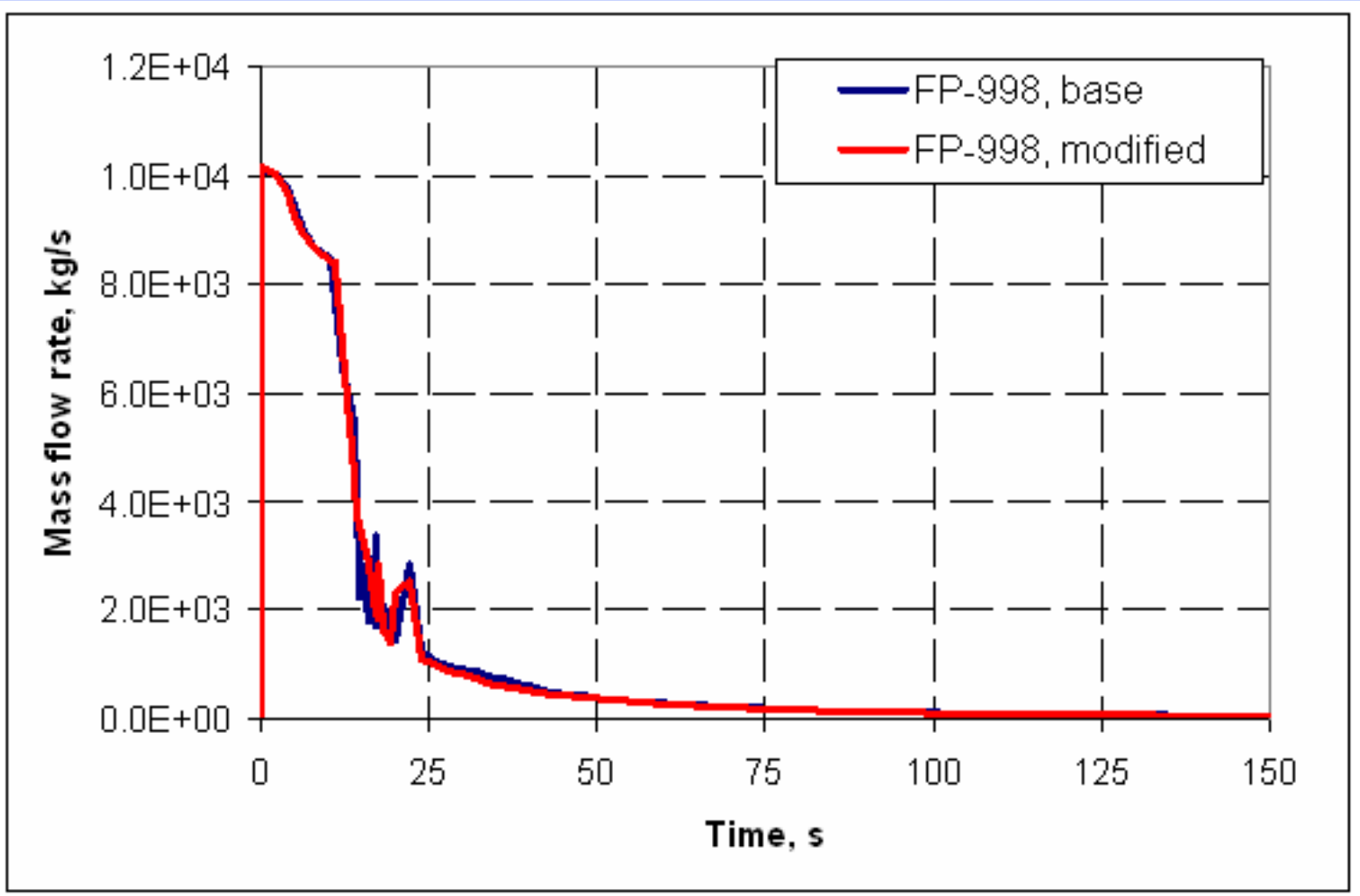
BWR large break test



BWR large break test: reactor vessel pressure



BWR large break test: mass flow rate through the break



Code refactoring

Eliminating of the code bottlenecks to improve code robustness and debugging capabilities:

- Removing of internal ("contains") subroutines (realized for CVH and RN1 packages).

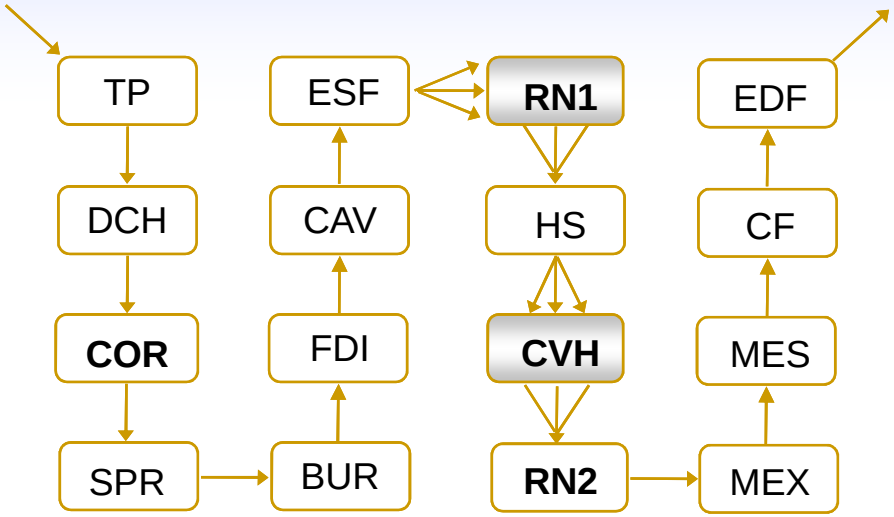
- With MELCOR 2.1 at the end of a MELCOR cycle, data are copied from the new to old arrays. Similarly, there are copies of data from old to new arrays in case of fallbacks. In MELCOR 1.8.6 there was no copy of data, but instead a pointers were switched. This "state rotation" of variables is preferred to copying of data and will improve performance, particularly for large data structure.

Parallelization of CVH package using OpenMP

- Partial parallelization approach
- Code modifications
- Changes in user input and output
- Testing

Partial parallelization approach

MELCOR time step



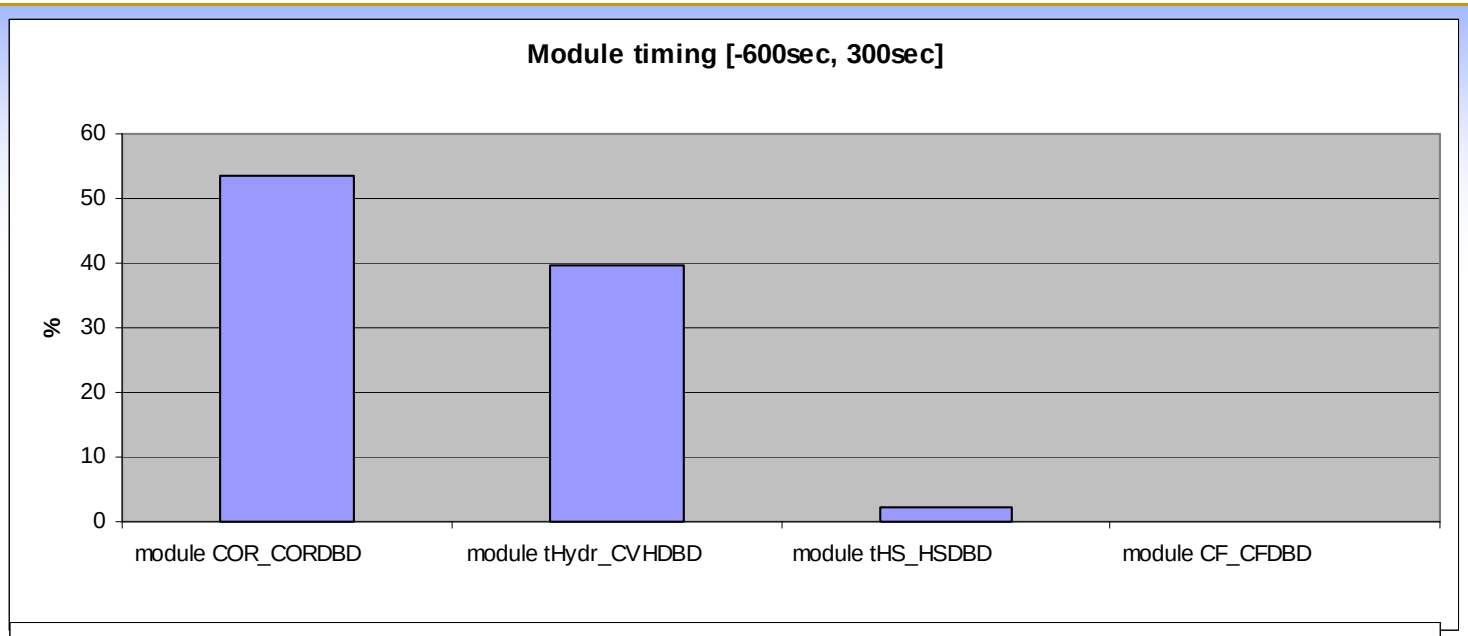
Each module has independent inner parallelization

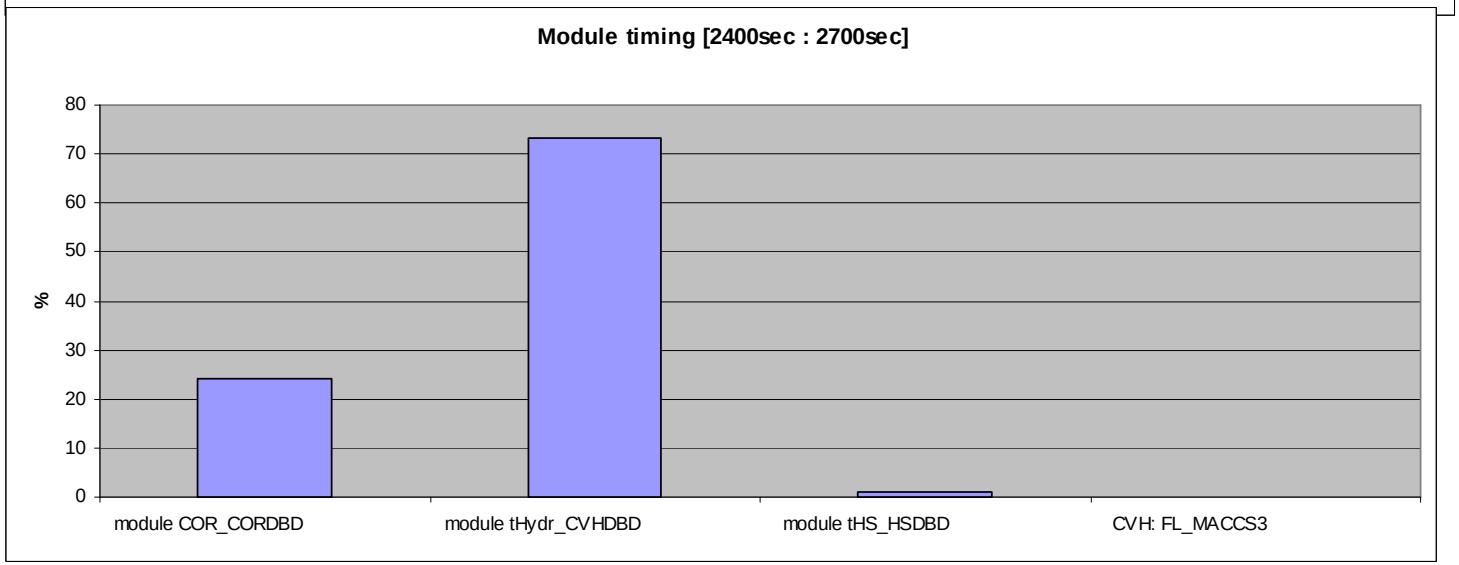
- Easier to localize problems
- Freedom of implementation

Investigation of code execution

- Performance measurement module (CR_TIMER_SET)
  - High-resolution measurements of multiple timing channels
  - Negligible small influence on the overall performance time (5000 calls to the timing functions per time step showed decrease of performance less than 1%)
  - The channels are printed in the output table in the same order they were called for the first time
  - *MPI_WTime* function (in case of MPI) or *OMP_get_wtime* (in case of OpenMP) is used for the high resolution (at least to microseconds) wall-clock time measurement

Investigation of code execution: TMI





Changes in the user input and output

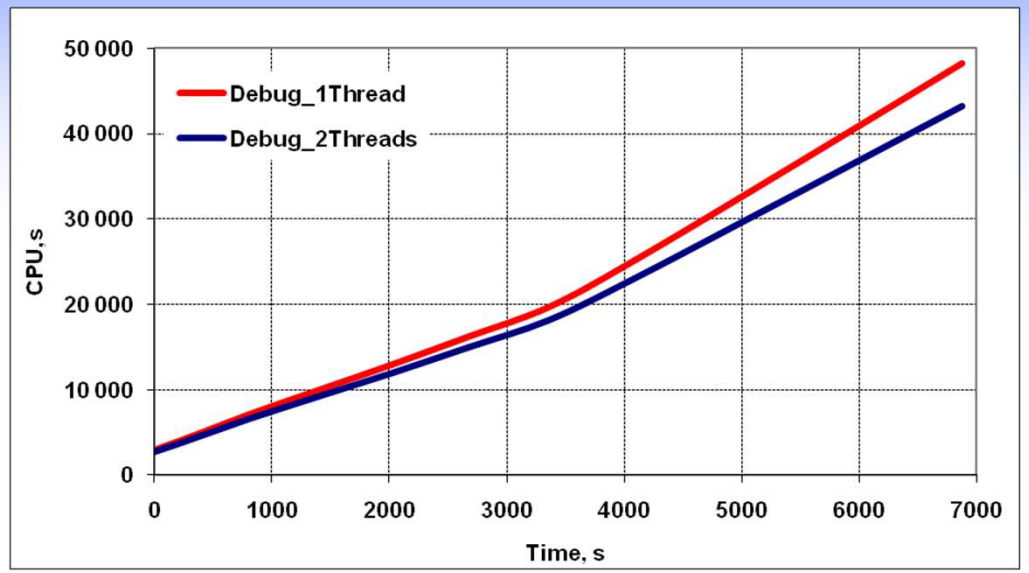
- Number of threads can be changed from the command line
  - NT – The flag for solver type (default NT=1)
  - Example

Melcor.exe *.inp NT=4

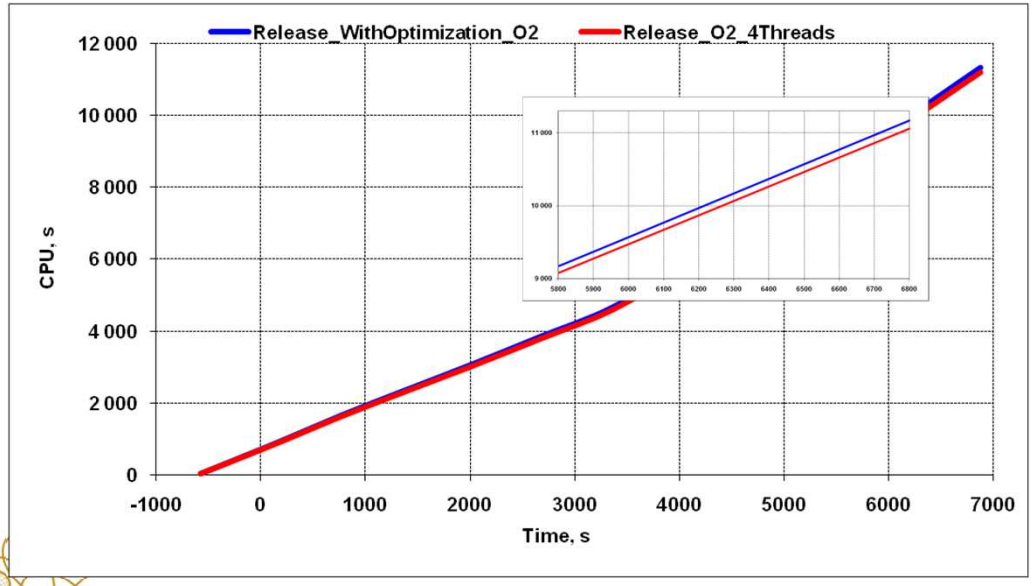
- New line showing the parallel number of threads used has been added to the output file

Parallel number of threads is set by user to 4

Testing: TMI



- Total CPU time is **13 hrs**.
- Parallelization (**2 threads**) gives an advantage **in 1.5 hrs**.



- Total CPU time is **3.1 hrs**.
- Parallelization (**4 threads**) gives an advantage in **2 minutes only**.

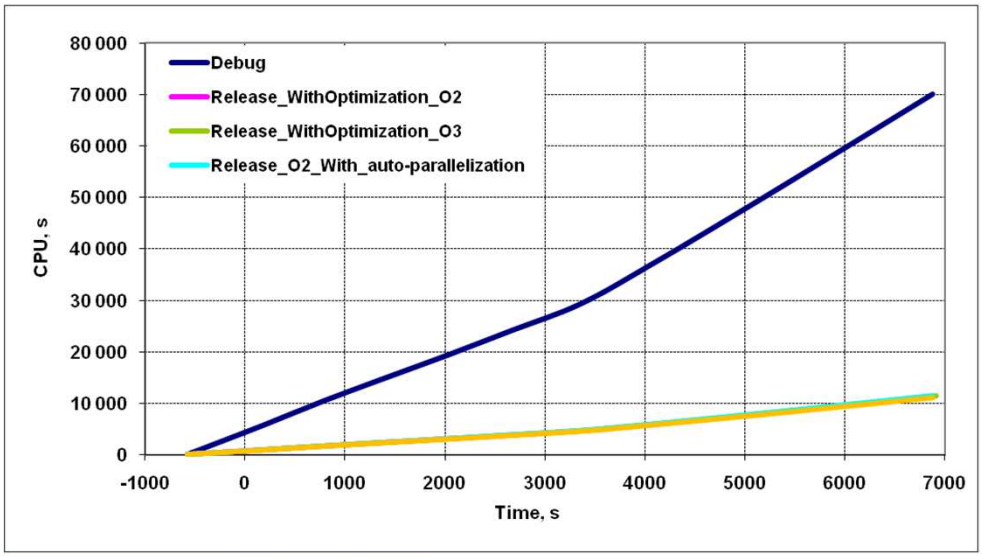
Compilation with optimization

- Two release versions have been built with O2 with O3

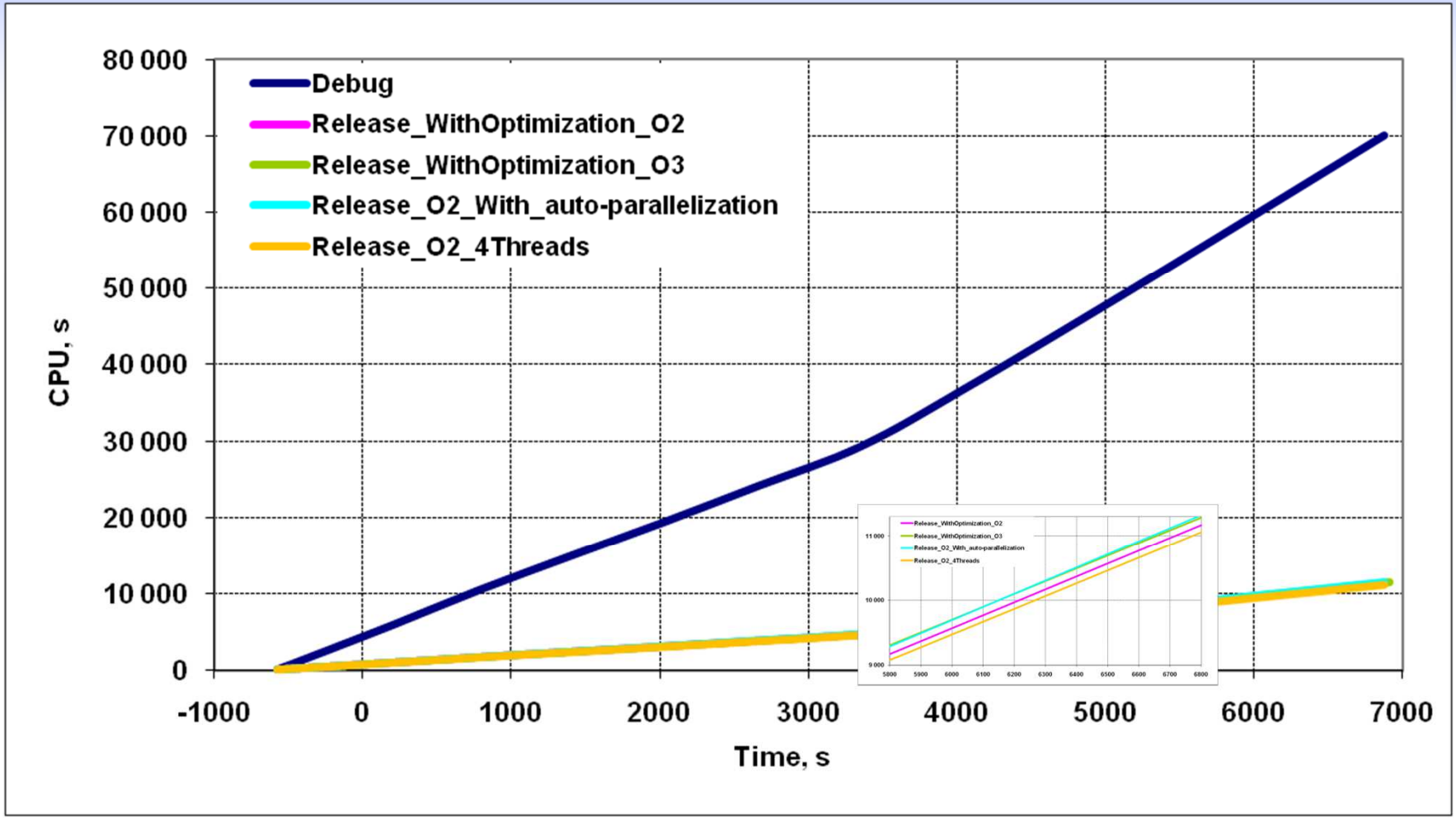
  Comparison for several representative tests (including TestLnew, BWR, PWR, TMI) has shown that release version is **5-6 times faster**

- A release version has been built with O2 and auto-parallelization

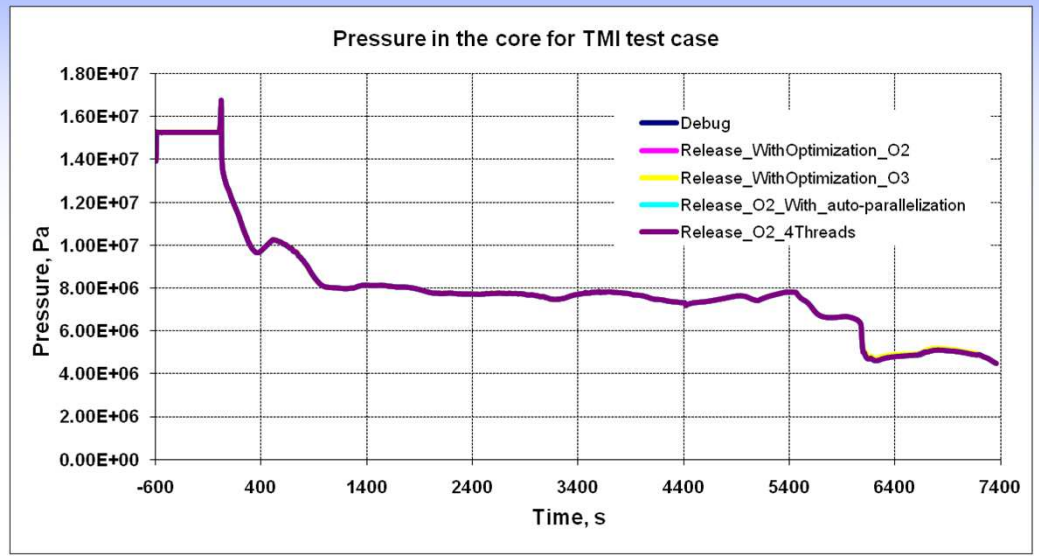
  Auto-parallelization give no CPU time advantage in comparison with release versions.

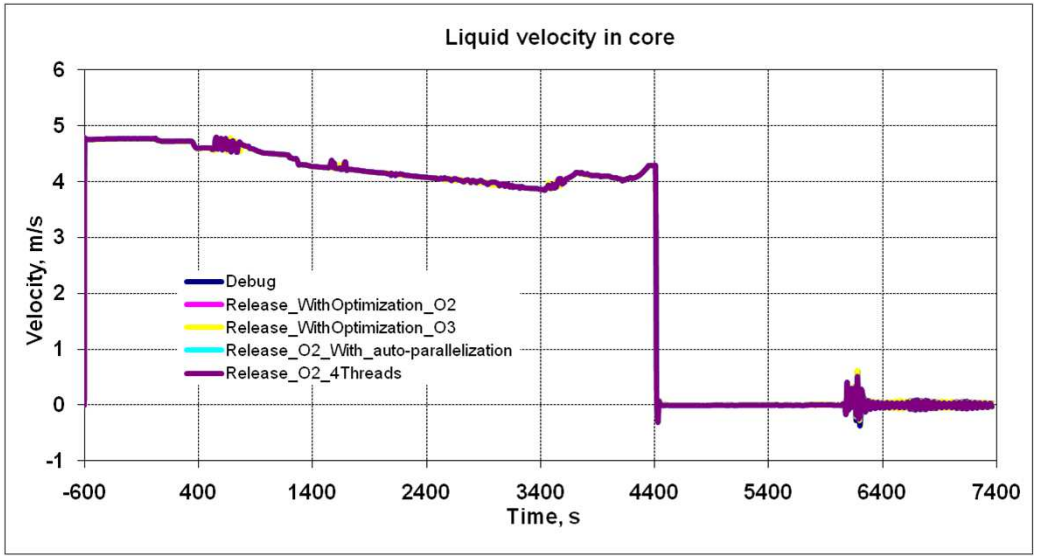


Performance tests: TMI

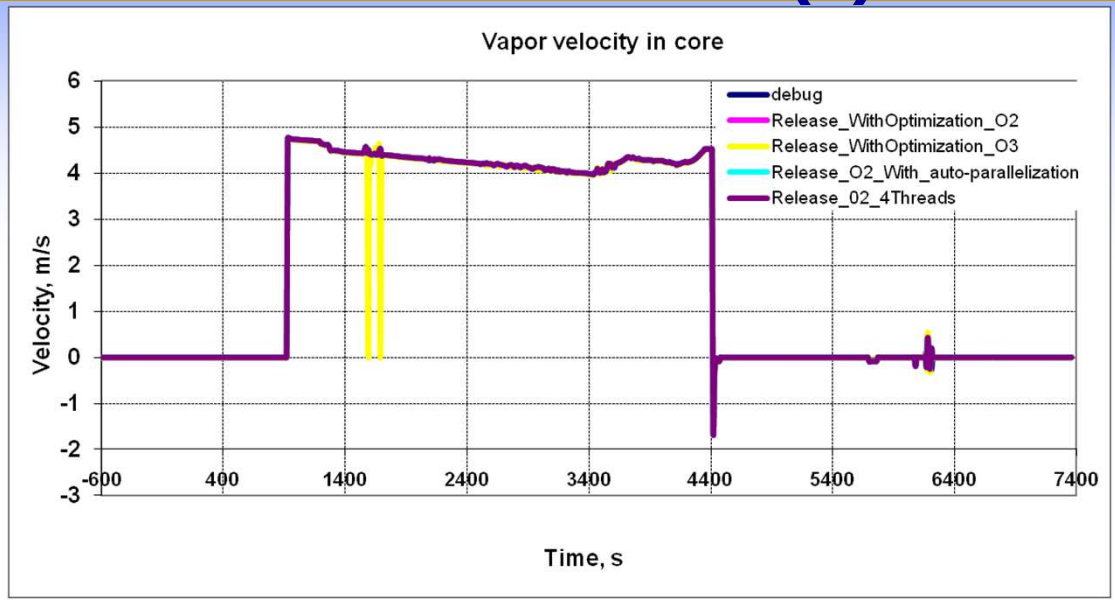


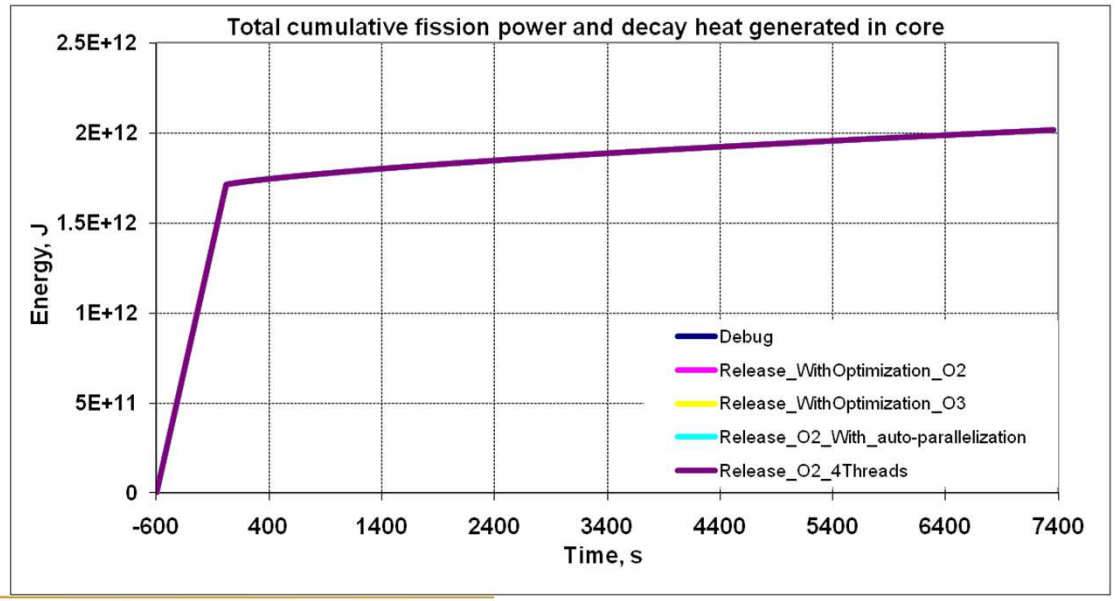
Performance tests: comparison of calculation results (1)





Performance tests: comparison of calculation results (2)





Conclusions

- Several ways of code performance improvement have been proposed:
  - Linear solver replacement
  - Critical flow model modernization
  - Partial parallelization with OpenMP
  - Code refactoring
- The modernizations have been implemented in MELCOR 2.1
- The debugging and wide testing is in process