РОССИЙСКАЯ АКАДЕМИЯ НАУК
безопасного развития атомной энергетики

RUSSIAN ACADEMY OF SCIENCES
Nuclear Safety Institute (IBRAE)

MELCOR Code Performance Improvement

I. Drobyshevskaya
N. Mosunova
A. Gorobets

EMUG Meeting, May 3, 2013
KTH Royal Institute of Technology,
AlbaNova, Stockholm, Sweden

Outline

- MELCOR code performance improvement strategies
- Variables swapping
- Parallelization
  - Package by package parallelization
  - MELCOR CVH/FL package parallelization results
  - MELCOR RN1 package parallelization results
  - MELCOR COR package parallelization
  - Performance and physical testing

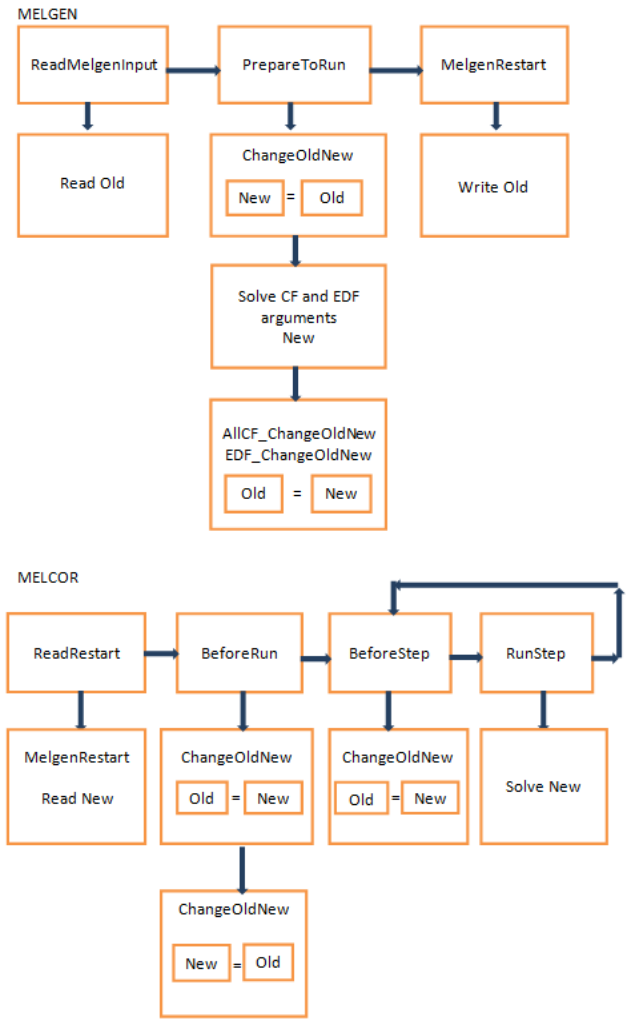
MELCOR Code Performance Improvement Strategies

- Parallelization
- Optimization
- Variables swapping instead of copying
- Numerical solvers modernization

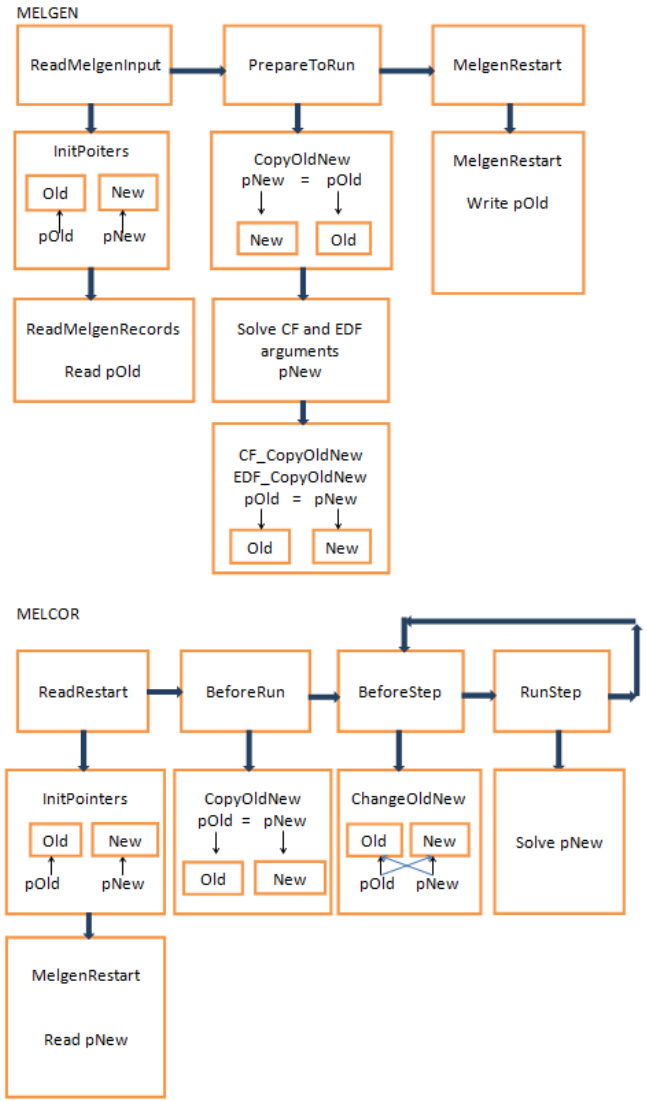
Percent of the Total CPU Time Spent on Old-New Data Copying Subroutines

Name	Ncycle	Copying(2795), % of the total CPU time	
		debug	release
TestLnew	10000	4.22682	3.46118
PWR	10000	4.07423	3.20576
BWR	10000	4.16238	2.64994
ISP37	84000	1.59107	1.20770
Accum	48000	4.69685	3.74983

Old-New Variables Copying Algorithm



Old-New Variables Swapping Algorithm



Testing Results

Name	Ncycle	Copying release		Swapping release	
		Time, s	% of CPU	Time, s	% of CPU
TestLnew	10000	6.02339	3.46118	5.63890	2.67194
PWR	10000	5.69175	3.20576	5.10419	2.50900
BWR	10000	5.73897	2.64994	5.06017	2.23071
ISP37	84000	8.52898	1.20770	4.52262	0.68059
Accum	48000	173.04676	3.74983	132.36028	2.49887

Testing Results (2)

Name	Ncycle	Copying (2795)		Swapping (4600)	
		release		release	
		Time, s	%	Time, s	%
GrandGulf_STSBO	36000	95.96534	2.0154	80.96818	1.9941
Zion_SBO	108000	286.7392	2.3668	257.9185	2.4085
TMI	274000	330.8196	1.4221	292.4056	1.3061
Surry_LBLOCA_MACCS	125000	432.7987	3.7648	308.4119	2.5111

Variables Swapping Final Remarks

- The old-new variables swapping algorithm has been realized for all of the MELCOR packages.
- The code modifications have been tested on the full set of shorter and longer runs.
- The CPU time decreasing for new swapping algorithm in comparison with old data copying algorithm for all input decks both in debug and release configurations.
- The bigger total CPU time is, the more efficient data swapping algorithm is.

Possible Strategies of Code Parallelization

- MPI, geometric parallelism
  Clusters, distributed memory model

- OpenMP, multithreading
  SMP systems, shared memory model

- Hybrid MPI + OpenMP
  Clusters with multi-core SMP nodes

MPI versus OpenMP

The Message Passing Interface (MPI) vs. Open Multiprocessing (OpenMP)

- MPI requires additional communication routines, additional data structures, partitioning hence it is more complex to implement.
- Partial parallelization in MPI requires additional scattering and gathering routines while with OpenMP we can spread threads wherever we want.
- MPI needs data transmission while OpenMP does not.

Parallelization Approaches in Case of MELCOR

Advantages of MPI

- Scalability – it is good to solve large problems on large computer systems
- Work on SMP systems as well as on distributed memory clusters

Disadvantages of MPI when applied to MELCOR

- Scalability is not needed – there is no big problems to solve. Computing load is due to big amount of time steps
- There is OpenMP to run on SMP systems which is much easier to use

SCs for Package by Package Parallelization

- The new sensitivity coefficients have been added to the MELCOR code to enable parallelization by packages.
- The SCs allow to switch on/off the parallelization of CVH or RN1 packages or set different number of threads for them.
- The priority of the SC is higher than the command line parameter NT.

CVH: Sensitivity Coefficient SC4420

4420 – Criteria for activation and deactivation of CVH package OpenMP parallelization

These coefficient is used to specify the number of threads used for CVH package calculations.

(1) - The number of threads.

0, The default number of threads for CVH package: one thread or the number given by command line argument NT.

1, The CVH/FL package run in parallel mode on one thread.

n, Sets the number of threads for CVH package equal to n, where n is the number from 2 to Parallel_MaxThreads

(default = 0.0, units = dimensionless)

RN: Sensitivity Coefficient SC7004

7004 – Criteria for activation and deactivation of RN1 package OpenMP parallelization

These coefficients are used to specify the number of threads used for RN1 package calculations.

(1) - The number of threads.

0, The default number of threads for RN1 package: one thread or the number given by command line argument NT.

1, The RN1 package run in parallel mode on one thread

n, Sets the number of threads for RN1 package equal to n, where n is the number from 2 to Parallel_MaxThreads

(default = 0.0, units = dimensionless)

Example of SCs Usage

```
!* Block: CVH melcor data
**************************************************
CVH_INPUT
CVH_SC 1  !SCnumber  Value    Index
          1    4420       1.0          1
!* END CVH
**************************************************
!* Block: RN1 melcor data
**************************************************
RN1_INPUT
RN1_SC 1 !SCnumber  Value    Index
          1    7004       4.0          1
!* END RN1
**************************************************
```

Changes in the Output File

These output messages are written in the output and diagnostic files.

```
     MODIFIED SENSITIVITY COEFFICIENT ARRAY SC4420 (CVH
Package Parallelization)
     ELEMENT(1): Number of Threads
     OLD VALUE =     0.000000E+00      NEW VALUE =
1.000000E+00

     MODIFIED SENSITIVITY COEFFICIENT ARRAY SC7004 (RN1
Package  Parallelization)
     ELEMENT(1): Number of threads
     OLD VALUE =     0.000000E+00      NEW VALUE =
4.000000E+00
```

Limitations of parallel usage

- Currently the maximum number of parallel threads is set to 8

- Attempt to set more then 8 threads will lead to the error message and stop the execution

```
Diagnostics during MELCOR input processing CVH package:
!(
Parallel_Set_CVH_Nthreads: Number of Threads        9
 exceeds Parallel_MaxThreads=         8
!)
```

MELCOR Timing

- The special timing module has been implemented into the MELCOR code.
- Three packages concentrate most of the time consumption of the code:
  - The Control Volume Hydrodynamics – Flow Path package (CVH/FL)
  - RadioNuclide Package (RN)
  - Core Package (COR)

RN1 Package Timing on NPP Inputs

- **Jelly**

CVH input/FL input **–123 control volumes and 254 flow paths.**

HS input **– 143 heat structures.**

COR input **– 17 axial levels and 6 radial rings.**

RN input **– 18 RN classes.**

- **ZionLBLOCA**

CVH input/FL input **–123 control volumes and 211 flow paths.**

HS input **– 220 heat structures.**

COR input **– 19 axial levels and 6 radial rings.**

RN input **– 16 RN classes.**

- **Surry_LBLOCA_MACCS**

CVH input/FL input **–142 control volumes and 263 flow paths.**

HS input **– 324 heat structures.**

COR input **– 17 axial levels and 6 radial rings.**

RN input **– 19 RN classes.**

- **FPT1**

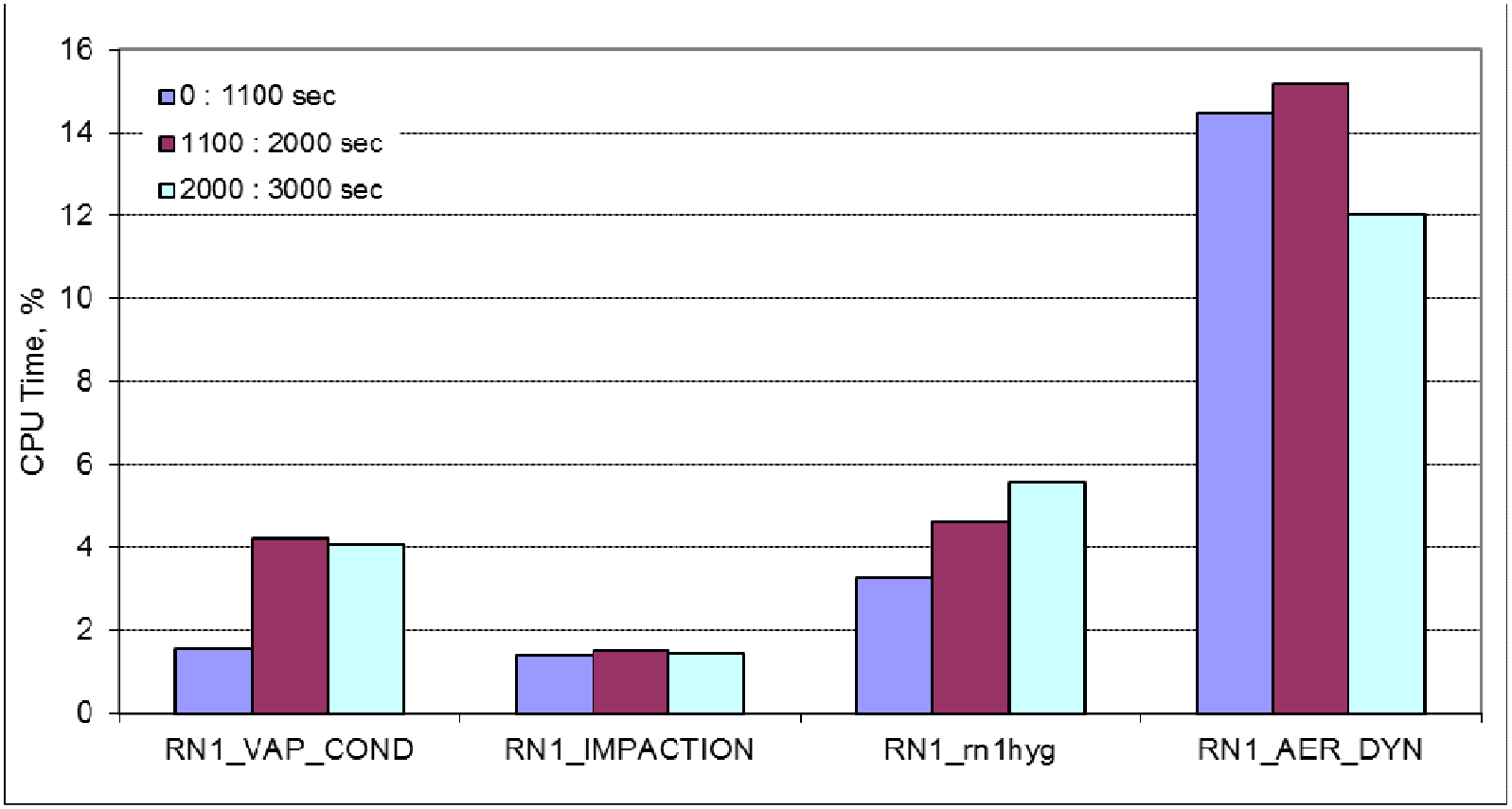
CVH input/FL input **– 31 control volumes and 29 flow paths.**

HS input **– 68 heat structures.**

COR input **– 31 axial levels and 2 radial rings.**

RN input **– 16 RN classes.**

Distribution of Computational Load for ZionLBLOCA Test Case



RN1 Package Performance Testing

Efficiency

Speedup: $S = t_1/t_P$

Efficiency: $E = S/P * 100\%$

t_1 – computational time in sequential mode

t_p – computational time on P cores

Efficiency of parallelization and speedup on 4 threads in release configuration

	FP1 (31 CVs)		Jelly (123 CVs)		ZionLBLOCA (123CVs)		Surry (142 CVs)	
	Eff, %	Speed Up	Eff, %	Speed up	Eff, %	Speed up	Eff, %	Speed up
VAP_COND	45.7	1.8	51.5	2.0	57.7	2.3	65.4	2.6
IMPACTION	32.6	1.3	39.3	1.6	57.7	2.3	61.6	2.5
Rn1hyg	35.3	1.4	73.9	3.0	60.9	2.4	50.6	2
RN1_AER_DYN	59.7	2.4	79.7	3.2	88.1	3.5	85.2	3.4

RN1 Package Parallelization Conclusions

- The most time consuming subroutines of the RN1 package has been parallelized using OpenMP, which lead to a speed up about 1.1 – 1.4 times on the four cores in release configuration
- The efficiency of parallelization depends on the number of CVs in the test case running.
- The efficiency of parallelization in release configuration is about 70% or higher for test cases with more than 100 CVs

CVH/FL Package Parallelization Results

The performance results running on two threads presented below.

Test case	Release 1 Thread, sec	Release 2 Threads sec
TMI	9056.5736	8802.6656
GrandGulf_STSBO	2906.8241	2897.8925
GrandGulf_LBLOCA	694.5123	693.67263
ATMI	17727.747	17253.602
Zion_sbo	2988.1562	2990.3781
ZionLBLOCA	10847.344	34763.69

CVH/FL Package Parallelization Conclusions

- The speedup of the whole code in debug configuration is about 1.5 – 2 times
- As most time consumption is concentrated in not parallelized linear solver the speedup of the whole code is negligible in release configuration
- Parallelization of the linear solver is unreasonable since the dimension of the matrix of the system of linear equations is to small to be efficiently parallelized
- To speedup the solver it was replaced. Performance testing of the new solver shown the speedup about 3 – 4 times

COR Package Timing

- Timing on the same set of NPP tests as RN1 package
- Several subroutines concentrated more than 1% of total CPU time:
  - COR_CORRN1
  - COR_CORRN3
  - COR_CORRN4

COR Package Parallelization

Distribution of computational load in COR_CORRN1 subroutine

Subroutine	CPU Time, %	Subroutine	CPU Time, %
COR: COR_CORRN1 1	0.01751	COR: CORTKE	0.00129
COR: COR_CORRN1 2	0.04961	COR: COR_CORCND	0.08271
COR: COR_CORDHC	0.8514	COR: COR_DO120	0.68517
COR: COR_CORSRC	0.02014	COR: COR_DO400	0.0027
COR: COR_CORPOW	0.00222	COR: COR_CORSTF	0.00632
COR: COR_CORCTK	0.04237	COR: COR_CORfzs	0.00107
COR: COR_CORGAP	0.00163	COR: COR_CORLHR	0.00037
COR: COR_CORRDS	0.04725	COR: COR_CORLHD	0.32962
COR: COR_CORM	0.05741	**COR: COR_COROXX ,OXY**	**2.93385**
COR: COR_CORRAD	0.20165	COR: COR_CORGEO	0.13195
COR: cor_corsub	0.01084	COR: COR_DO260	0.24714
COR: COR_CORMPS	0.0127	COR: COR_CORBAL	0.22954
COR: COR_CORTSV	0.40246	**Sum:**	**6.36892**

COR Package Parallelization Conclusions

- The study of distribution of computational load of the COR package has shown only several sources which take at least 1 – 3 percent of total CPU time

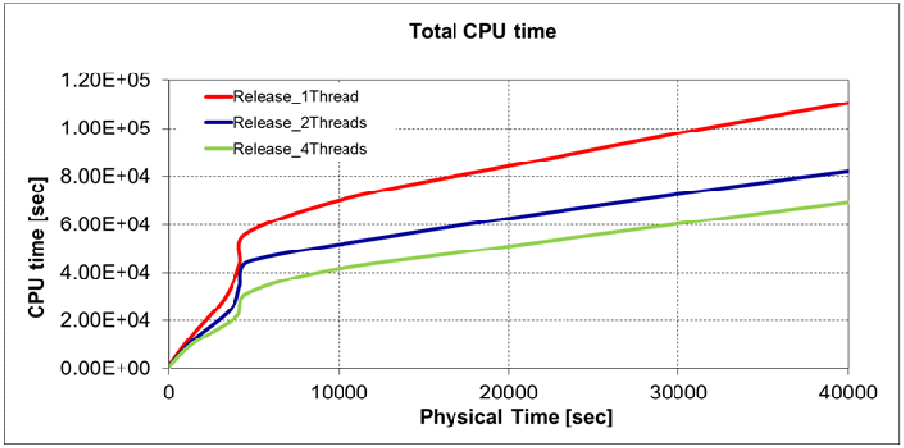
- Most of the time consumption is concentrated in plenty of small sources 0.5 – 0.01 percent of total CPU each

- The detailed study of most loaded places shown the interdependencies preventing from correct parallelization

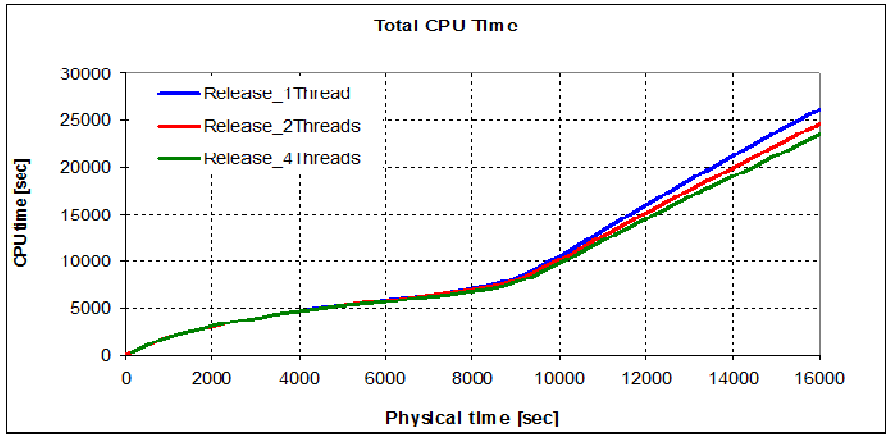
Performance Results

- RN1 and CVH/FL in parallel mode
- 4 parallel threads
- Release configuration

	FPT1	Jelly	Zion LBLOCA	Surry	Vanam
Total Speedup	1.12	1.43	1.61	1.59	1.63



SURRY_LBLOCA_MACCS test case

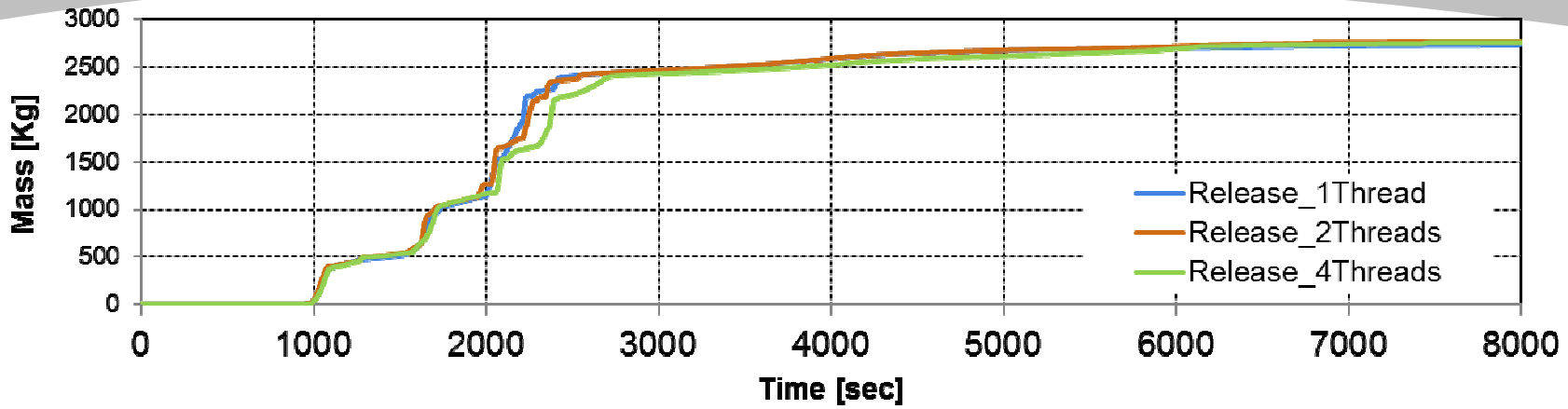


FPT1 test case

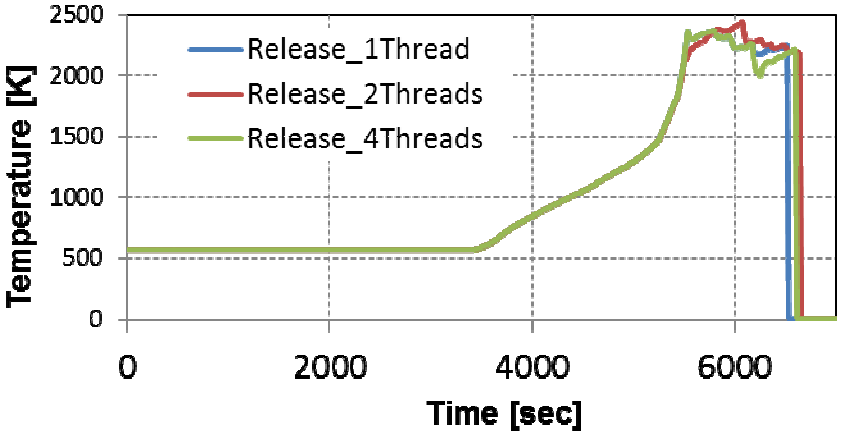
Comparison of Physical Results

- All changes in the code have been tested on the big set of tests including the chosen NPP tests in debug and release configurations
- The results are identical running in sequential mode
- In parallel mode the results are either identical or have negligible differences in both debug and release configurations

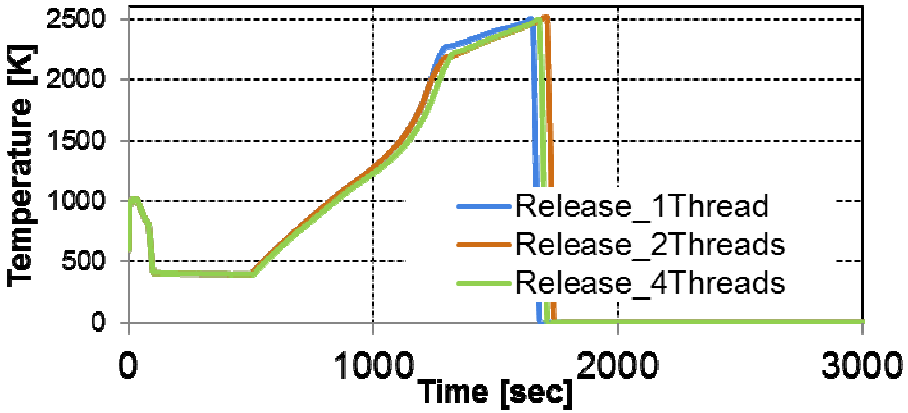
Comparison of Physical Results (2)



Total non-radioactive plus radioactive mass released from COR components for SURRY_LBLOCA_MACCS



Temperature of the cladding for Jelly test case



Temperature of the cladding for SURRY_LBLOCA_MACCS test case

Conclusions

- Several ways of MELCOR code performance improvement have been developed
- The parallelization of two time consumption modules has been accomplished
- The swapping algorithm has been realized instead of copying the old/new data for most of the MELCOR packages. The total CPU time decrease for all the input decks tested