Testing, Simulating, and Profiling Designated Problems for CASL

Yael M. Camacho Bonaparte
Master of Engineering in Computer Engineering
Dr. Luis Vicente
Electrical & Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— Simulation is an essential step for future implementations of more efficient and safer reactor components. In nuclear Technology, this translates to cost, fuel, and waste reduction, and enhanced safety. The software Coolant Boiling in Rod Arrays – Two Fluid (COBRA-TF) is currently undergoing development and until now has not been profiled to measure its performance. This project presents profiling data and data analysis of the code's performance. We profiled two different version of the software while simulating two different input files. The profile data extracted from the simulations provided developers a better understanding of the code's behavior. The results of this project will help developers improve the code's performance in future versions by identifying the most time consuming portions of the code. The project involved activities such as testing, simulation, running designated problems on different programs or environments, analyzing, and verifying the results, and providing feedback to the development team.*

Key Terms *— Optimization, Profiling, Simulation, Virtual Nuclear Reactor.*

BACKGROUND

Scientists have been modeling real-life scenarios to predict system behavior of different parameters. Modeling is a cheaper, safer, and faster way to understand how systems. Dangerous system conditions can be simulated in a totally safe environment in order to determine the real-life results.

The Consortium for Advanced Simulation of Light Water Reactors (CASL) is a partnership between the Department of Energy (DOE) laboratories, such as Oak Ridge National Laboratory, Los Alamos National Laboratory, Idaho National Laboratory, and Sandia National Laboratories, highly qualified educational institutions, such as University of Michigan, North Carolina State University, and Massachusetts Institute of Technology, and outstanding industry companies such as Electric Power Research Institute, Tennessee Valley Authority, and Westinghouse. By implementing advanced modeling and simulation capabilities, CASL plans to reduce costs, fuel consumption, and waste while increasing safety.

COBRA-TF is a thermal-hydraulic sub channel code for light water reactors transient analyses developed by the Reactor Dynamics and Fuel Management Group (RDFMG) and distributed to Oak Ridge National Laboratory (ORNL) for the CASL Project. CASL is using COBRA-TF to develop a modeling tool called the Virtual Environment for Reactor Applications (VERA). This tool will simulate the performance of light water reactors using current and advanced modeling and simulation capabilities.

SIGNIFICANCE OF THE STUDY

Large complex programs often have pieces of code with poor performance [1]. This can be caused by several factors, being the coding techniques used the most common factor. This can be improved by using different profile tools. Profiling helps identify the most time-consuming portion of the program.

The ability to profile COBRA-TF will provide developers useful information about the code's performance. This information will allow developers learn where COBRA-TF is spending most of the execution time, which functions were called by whom, and how many times each function or routine ran. This profiling information will assist developers to optimize COBRA-TF in future versions.

Future optimizations of COBRA-TF will allow the program to execute faster and more efficiently. Scientists will benefit from these optimizations since they will be able to run simulations faster which mean less execution time and power consumption.

OBJECTIVES

The objective of the project is to profile execution of COBRA-TF in order to understand its behavior, find out where the time is being spent, which functions are the most time consuming ones, analyze and visualize the results in order to provide developers a clear performance report of the simulations. This report will help them improve the program's performance in future versions.

TEST BED METHODOLOGY

A computer with Linux (OpenSUSE) installed was used for this project. Two versions (V1.0 and V1.1) of COBRA-TF were profiled in order to compare and contrast the results between them. Two input files were simulated in each version of COBRA-TF in order to compare the results. These are a small input file (Input I) of about 9KB and a relatively large input file (Input II) of about 210KB. The input files and the type of simulation performed were provided by CASL. Table 1 shows the configuration for each simulation.

Table 1
Test Bed Configurations

Configuration Name	COBRA-TF version	Input file version
Test bed I	1.0	Input I
Test bed II	1.1	Input I
Test bed III	1.0	Input II
Test bed IV	1.1	Input II

Four simulations were profiled during this project using the test beds shown at table 1. After each run, the profile gprof output data were saved in a safe location to be analyzed later on. After collecting all the profile data, the information was trimmed and organized in a single document, the call graph profiles were visualized with kprof and graphviz and the information was presented to the developers.

Test Bed I

Test bed I was executed after compiling and linking the code with the profile option "-pg" enabled [4]. The simulation took about twelve minutes to complete.

Table 2
Flat Profile from Test Bed I

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	name
26.71	26.28	26.28	17577861	tgas
22.20	48.12	21.84	17677	xschem
9.67	57.64	9.52	2669227	intfr
6.18	63.72	6.08	17677	temp
6.07	69.69	5.97	2669227	boiling
4.27	73.89	4.20	2651550	reduce
3.71	77.54	3.65	17677	heat
3.26	80.75	3.21	17685	post3d
2.42	83.13	2.38	9605533	prop
2.12	85.22	2.09	2651550	bacout
2.08	87.27	2.05	10664669	sat
2.08	89.32	2.05	2669227	veloc
1.57	90.86	1.54	2651550	fillro
1.09	91.93	1.07	9605533	transp
1.08	92.99	1.06	4266172	hgas
…	…	…	…	…

Table 2 shows the profile output data for the top 99% time consuming of test bed I.

Table 3
Call Graph Profile from Test Bed I

granularity: each sample hit covers 4 byte(s) for 0.01% of 98.40 seconds

index	% time	self	children	called	name
[1]	100	0.00	98.40		MAIN [1]
		0.77	97.62	1/1	trans [2]
		0.00	0.01	1/1	input [36]
		0.00	0.00	1/1	init [50]
		0.00	0.00	1/1	blkdat [48]
		0.77	97.62	1/1	MAIN [1]
[2]	100	0.77	97.62	1	trans [2]
		0.00	50.89	17677/17677	outer [3]
		0.24	31.91	17677/17677	prep3d [5]
		3.21	11.34	17685/17685	post3d [10]
		0.00	0.02	6/6	edit [33]

		0.01	0.00	17707/17707	timstp [38]
		0.00	0.00	30/30	dmpit [43]
		0.00	50.89	17677/17677	trans [2]
[3]	51.7	0.00	50.89	17677	outer [3]
		21.84	29.05	17677/17677	xschem [4]
		21.84	29.05	17677/17677	outer [3]
[4]	51.7	21.84	29.05	17677	xschem [4]
		9.52	5.53	2669227/2669227	intfr [9]
		1.54	4.20	2651550/2651550	fillro [13]
		3.96	0.00	2651550/17577861	tgas [7]
		2.05	0.00	2669227/2669227	veloc [17]
		0.78	0.00	2651550/2651550	xtra1 [21]
		0.51	0.00	2651550/10664669	sat [16]
		0.11	0.15	2651550/2651550	dvdh1 [27]
		0.22	0.00	17677/17677	gssolv [28]
		0.20	0.00	2651550/5320778	gasp [24]
		0.19	0.00	2651550/2651550	dvdpv [29]
		0.08	0.00	2651550/2651550	dvdhv [31]

Since the call graph profile is extremely large and complex, table 3 shows only the first four entries of the output file.

Test Bed II

Test bed II took about eight minutes to complete. It ran 33% faster than test bed I while executing with Input I. The Central Processing Unit (CPU) time used by test bed II was slightly bigger than test bed I, but it ran faster overall.

Table 4
Call Graph Profile from Test Bed II

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	name
30.49	31.18	31.13	17577861	tgas
23.47	55.18	24.00	17677	xschem
8.68	64.06	8.88	2669227	intfr
5.31	69.49	5.43	17677	temp
5.29	74.90	5.41	2669227	boiling
3.81	78.80	3.90	17677	heat
3.61	82.49	3.69	2651550	reduce
2.94	85.50	3.01	17685	post3d
2.12	87.67	2.17	9605533	prop
1.89	89.60	1.93	10664669	sat
1.86	91.50	1.90	2651550	bacout
1.72	93.26	1.76	2669227	veloc
1.56	94.86	1.60	2651550	fillro
1.23	96.12	1.26	9605533	transp
1.12	97.27	1.15	4266172	hgas
…	…	…	…	…

Table 4 shows the profile output data for the top 99% time consuming routines of test bed II.

Table 5
Call Graph Profile from Test Bed II

granularity: each sample hit covers 4 byte(s) for 0.01% of 98.40 seconds

index	% time	self	children	called	name
[1]	100	0.00	102.26		MAIN 1]
		0.69	101.56	1/1	trans [2]
		0.00	0.01	1/1	input [36]
		0.00	0.00	1/1	init [50]
		0.00	0.00	1/1	blkdat [48]
		0.69	101.56	1/1	MAIN 1]
[2]	100	0.69	101.56	1	trans [2]
		0.01	53.00	17677/17677	outer [3]
		0.21	32.73	17677/17677	prep3d [5]
		3.01	12.57	17685/17685	post3d [10]
		0.00	0.01	6/6	edit [33]
		0.00	0.01	17707/17707	timstp [38]
		0.00	0.00	30/30	dmpit [43]
		0.01	53.00	17677/17677	trans [2]
[3]	51.8	0.01	53.00	17677	outer [3]
		24.00	29.00	17677/17677	xschem [4]
		24.00	29.00	17677/17677	outer [3]
[4]	51.8	24.00	29.00	17677	xschem [4]
		8.88	6.28	2669227/2669227	intfr [9]
		1.60	3.69	2651550/2651550	fillro [13]
		4.70	0.00	2651550/17577861	tgas [7]
		1.76	0.00	2669227/2669227	veloc [17]
		0.57	0.00	2651550/2651550	xtra1 [21]
		0.48	0.00	2651550/10664669	sat [16]
		0.17	0.17	2651550/2651550	dvdh1 [27]
		0.28	0.00	17677/17677	gssolv [28]
		0.24	0.00	2651550/5320778	gasp [24]
		0.12	0.00	2651550/2651550	dvdpv [29]
		0.06	0.00	2651550/2651550	dvdhv [31]

Since the call graph profile is extremely large and complex, table 5 shows only the first four entries of the output.

Test Bed III

Test bed III took about five hours and thirty minutes to complete.

Table 6
Flat Profile from Test Bed III

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	name
45.55	706.09	706.09	1122	xschem
10.83	973.08	186.99	124634943	tgas
6.86	1091.52	118.44	1122	result_channel
5.92	1193.65	102.13	1122	temp
5.44	1287.61	93.96	45936	sstemp
4.27	1361.38	73.77	1122	result
3.53	1422.34	60.96	31416	intfr
2.54	1466.15	43.81	1122	heat

2.14	1503.00	36.85	10178784	boiling
2.04	1538.13	35.13	1122	gssolv
1.33	1561.04	22.91	1122	result_gap
1.04	1578.96	17.92	94462118	prop
1.02	1596.46	17.92	91613230	sat
…	…	…	…	…

Table 6 shows the profile output data for the top 99% time consuming routines of test bed III.

Table 7

Call Graph Profile from Test Bed III

granularity: each sample hit covers 4 byte(s) for 0.01% of 98.40 seconds

index	% time	self	children	called	name
[1]	100	0.00	1725.80		MAIN [1]
		2.29	1629.19	1/1	trans [2]
		0.00	94.32	1/1	input [14]
		0.00	0.00	1/1	blkdat [50]
		0.00	0.00	1/1	init [52]
		2.29	1629.19	1/1	MAIN [1]
[2]	94.5	2.29	1629.19	1	trans [2]
		0.00	1011.35	1122/1122	outer [3]
		3.16	309.39	1122/1122	prep3d [5]
		0.02	243.09	1123/1123	edit [7]
		16.11	46.08	1122/1122	post3d [20]
		0.00	0.00	1124/1124	timstp [45]
		0.00	0.00	2/2	dmpit [48]
		0.00	1011.35	1122/1122	trans [2]
[3]	58.6	0.00	1011.35	1122	outer [3]
		786.09	225.26	1122/1122	xschem [4]
		786.09	225.26	1122/1122	outer [3]
[4]	58.6	786.09	225.26	1122	xschem [4]
		60.96	57.00	31416/31416	intfr [11]
		35.13	0.00	1122/1122	gssolv [22]
		12.44	13.57	9815256/9815256	fillro [23]
		15.32	0.00	31416/31416	vdrift [26]
		14.73	0.00	9815256/9815256	tgas [9]
		10.69	0.00	31416/31416	veloc [30]
		1.88	0.00	9815256/91613230	sat [25]
		0.56	0.53	9815256/9815256	dvdhl [36]
		0.95	0.00	9815256/9815256	xtra1 [37]
		0.88	0.00	9815256/20357569	gasp [34]
		0.40	0.00	9815256/9815256	dvdpv [41]
		0.23	0.00	9815256/9815256	dvdhv [43]

Since the call graph profile is extremely large and complex, table 7 shows only the first four entries of the output file.

Test Bed IV

Test bed IV took about two hours and thirty minutes to complete. Test bed IV ran 55% faster than test bed III. The Central Processing Unit (CPU) time used by test bed IV was slightly smaller than test bed III. Note the improvement in running time although both test bed used similar CPU time.

Table 8

Call Graph Profile from Test Bed IV

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	name
45.48	765.52	765.52	1122	xschem
11.08	952.09	186.57	124634943	tgas
6.59	1063.09	111.00	1122	result_channel
5.84	1161.36	98.27	45936	sstemp
5.81	1259.22	97.86	1122	temp
4.23	1330.38	71.16	1122	result
3.52	1389.56	59.18	31416	intfr
2.54	1432.25	42.69	1122	heat
2.07	1457.11	34.86	10178784	boiling
2.04	1501.37	34.26	1122	gssolv
1.23	1522.04	20.67	1122	result_gap
1.01	1539.00	16.96	94462118	prop
1.00	1555.83	16.83	91613230	sat
…	…	…	…	…

Table 8 shows the profile output data for the top 99% time consuming routines of test bed IV.

Table 9

Call Graph Profile from Test Bed IV

granularity: each sample hit covers 4 byte(s) for 0.01% of 98.40 seconds

index	% time	self	children	called	name
[1]	100	0.00	1683.15		MAIN [1]
		2.18	1582.39	1/1	trans [2]
		0.00	98.57	1/1	input [14]
		0.01	0.00	1/1	blkdat [45]
		0.00	0.00	1/1	init [53]
		2.18	1582.39	1/1	MAIN [1]
[2]	94.1	2.18	1582.39	1	trans [2]
		0.00	987.61	1122/1122	outer [4]
		2.92	300.16	1122/1122	prep3d [5]
		0.00	230.20	1123/1123	edit [7]
		15.70	45.80	1122/1122	post3d [20]
		0.00	0.00	1124/1124	timstp [47]
		0.00	0.00	2/2	dmpit [50]
		765.52	222.09	1122/1122	outer [4]
[3]	58.7	765.52	222.09	1122	xschem [3]
		59.18	56.16	31416/31416	intfr [11]
		0.00	987.61	1122/1122	trans [2]
[4]	58.7	0.00	987.61	1122	outer [4]
		765.52	222.09	1122/1122	xschem [3]

Since the call graph profile is extremely large and complex, figure 9 shows only the first four entries of the output file.

RESULTS

COBRA-TF V1.0 and V1.1 were profiled simulating two different input files. For the purpose of this project the difference between V1.0 and V1.1 is limited to memory management improvements; any other specific detail is out of the scope of this project.

Test beds I and II provided information about simulations of input I from both versions of COBRA-TF. Comparing the information shown in table 2 and table 4, we can observe that CPU times used by test bed I and II were similar, 98.40 and 102.26 respectively. As shown in table 10, test bed II used about 4% more CPU time than test bed I, although it ran about 33% faster. This suggests that the performance gained between COBRA-TF V1.0 and V1.1 was probably due to better memory management.

Table 10
Contrast between Test Bed I and II

Configuration	CPU Time (s)	Time (min)
Test bed I	98.40	12
Test bed II	102.26 (3.7% more)	8 (33% faster)

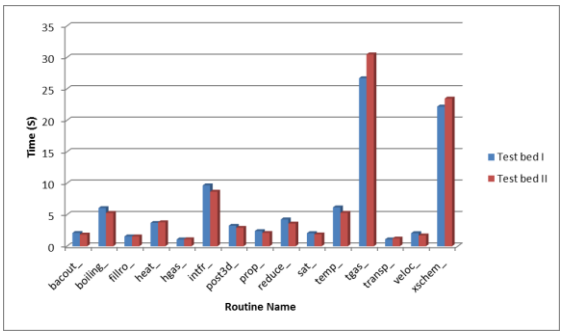

Figure 1
Self-Seconds Comparison between Test Bed I and II

The data were also presented to developers in bar plots, comparing the top 99% time-consuming routines between the two versions of COBRA-RF, as shown in Figure 1.

Test beds III and IV provided information about both simulations with input II. After analyzing the information shown in Figure 6 and Figure 8, we noticed that test bed IV used about 2.5% less CPU time than test bed III. During these particular simulations, we noted a large difference in the actual running time, 5.5 hours versus 2.5 hours (55% faster), as shown in table 11. Again, since the CPU time used on both simulations is similar, the improvement in performance seems to be accounted to memory handling improvements.

Table 11
Contrast between Test Bed III and IV

Configuration	CPU Time (s)	Time (hrs)
Test bed III	1725.80	5.5
Test bed IV	1683.15 (2.5% less)	2.5 (55% faster)

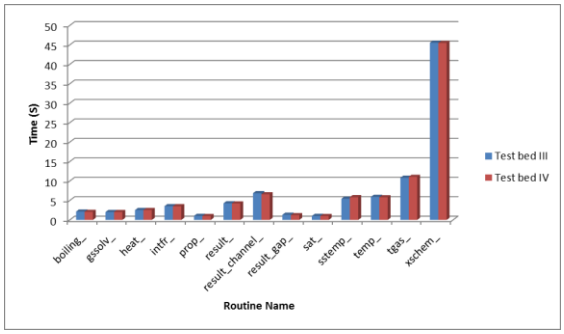

Figure 2
Self-Seconds Comparison between Test Bed III and IV

The data were also presented to developers in a bar plots, comparing the top 99% time-consuming routines between the two versions of COBRA-TF, as shown in Figure 2.

After analyzing the overall data regardless of which version of COBRA-TF was used and which input file was simulated, we found that most of the CPU time is being used by only two routines: tgas and xschem. “tgas” is a small routine that calculates vapor temperatures and specific heat capacity of vapors. “xschem” is a routine that linearizes the momentum, continuity, and energy equation. During the four simulations, the CPU times spent in these two routines were 48.91%, 53.96%, 56.38%, and 56.56 respectively. In average, the total CPU time spent on these two routines was about 53.95%. It is likely to obtain better performance by implementing these two functions.

These two routines behave in an entirely different way but are the top two time consuming

routines. Tgas is a remarkably small routine that runs extremely fast. The cause for the high CPU time used by this routine is that it is called about 124.6 million times in simulation IV as shown in Figure 8. Since it is a small and fast routine it is not optimal to implement the routine itself to obtain better performance. It may be better to optimize other functions that call this routine in order to reduce the calls count. By calling it less, the overall code's performance may improve. On the other hand, the xschem routine is not called too many times. This routine is still between the top two CPU time consuming routines. This can be due to the circumstance that it has many sub-routine calls, as shown in Figure 1. CPU time improvement may be obtained by implementing the sub-routines that are called by it.

FUTURE WORK

This project can be implemented in various ways, the most prominent being to extend the profiling techniques to memory usage. There are many tools available to profile memory usage during execution, such as valgrind and mprof, which can measure memory usage and detect memory leaks during execution. Tracing memory leaks facilitates the removal of memory request/release errors in C++ programs [7]. Also, they are used to study the dynamic memory allocation behavior of programs [5]. Memory profiling may lead to substantially improvement of the overall performance of COBRA-TF since only a small portion of the actual time spent executing was CPU time.

ACKNOWLEDGMENTS

The project was supported by the Research Alliance in Math and Science program (RAMS). RAMS is sponsored by the Office of Advanced Scientific Computing Research, U.S. Department of Energy. The work was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. This work has been authored by a contractor of the U.S. Government, accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

Special thanks to my mentor John Turner (ORNL) for all his guidance and support during this project.

REFERENCES

[1] Felanson, J, et al., "GNU grpof: the GNU profiler", *Free Software Foundation Inc.*, 1988.

[2] Knuth, D, "An Empirical Study of FORTRAN Programs", *Software: Practice and Experience*, 1971.

[3] Graham, S, *et al.*, "An Execution Profiler for Modular Programs", *Software: Practice and Experience*, 1983.

[4] Graham, S, *et al.*, "Gprof: A Call Graph Execution Profiler", *ACM Sigplan Notices*, 1982.

[5] Zorn, B, *et al.*, "A Memory Allocation Profiler for C and Lisp Programs", *Proceedings of the Summer USENIX Conference*, 1988.

[6] Pettis, K, *et al.*, "Profile Guided Code Positioning", *ACM Sigplan Notices*, 1990.

[7] Beaty, S., "A technique for tracing memory leaks in C++", *ACM Sigplan*, 1994.

[8] Shende, S, "Profiling and Tracing in Linux", *Proceedings of the Extreme Linux Workshop*, 1999.

[9] Serrano, M, *et al.*, "Understanding Memory Allocation of Scheme Programs", 2000.

[10] Avramova, M, *et al.*, "Improvements and Applications of COBRA-TF for Stand-alone and Coupled LWR Safety Analyses", *Proceedings: PHYSOR*, 2006.

[11] Bradley, B, *et al.*, "Automatic Memory Leak Detection", *U.S. Patent No. 20-130-054-923*, 2013.

[12] Reactor Dynamics and Fuel Management Group, www.mne.psu.edu/rdfmg, *last accessed*, July 2013.

[13] Consortium for Advanced Simulations of LWRs, www.cals.gov, last accessed, July 2013.

[14] Oak Ridge National Laboratory, www.ornl.gov, last accessed, June 2013.

[15] Kprof, www.kprof.sourceforge.net, last accessed, June 2013.