ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPWI020

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2015). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

DEVELOPMENT OF SIMPLE TRACKING LIBRARIES FOR ALS-U*

H. Nishimura#, D. Robin, K. Song, C. Steier, C. Sun, W. Wan
LBNL, Berkeley, CA 94720, USA

Abstract

A conceptual lattice design study of a new diffraction-limited light source requires complex and numerically intensive calculations due to increasing number of fitting parameters. This paper reports our ongoing effort of upgrading accelerator modeling and simulation libraries to carry out such design studies efficiently.

THE TRACY LIBRARIES

Tracy[1] is an accelerator modeling and simulation code developed as a part of the Advanced Light Source (ALS) conceptual design study[2] in late 1980's. The original version was written in Pascal, and its accelerator library was extracted and rewritten in C/C++ and later in C#.

The C/C++ version of the library was called Goemon, which is now Tracy++[3]. One of its applications is the use with multi-objective genetic algorithms (MOGA)[4] to optimize the existing ALS lattice[5], and the new lattice for a diffraction-limited light source called ALS-U[6] by using OpenMPI[7].

It is not common to use C# for scientific computing yet, however, the development effort of Tracy was moved to the C# version called Tracy#[8] a decade ago for better development efficiency and horizontal integrability than the C/C++ version but with some performance penalty at run time.

Taking the design study of ALS-U as an opportunity, the upgrade of these Tracy libraries have started. Cleaning up and simplifying the internal structures, they are upgraded to take benefit of the modern software and hardware technologies. These new version is called Tracy.Lite that comes in both C++ and C# versions.

TRACY.LITE

Goals

The goal is to create simpler, faster, and more reliable and flexible libraries than the Tracy++ and Tracy# combination. We do this first by limiting our scenario to the first phase of the conceptual lattice design studies where lattice errors are not considered but many parameter optimizations with multiple objectives and also straightforward scans are required. The modeling of realistic lattice errors is for the second phase.

Tracy.Lite supports multiple usage modes; a highly-parallelized batch execution mode for both MOGA and straightforward scan, and interactive mode using scripts.

*Work supported by the U.S. Department of Energy under Contract No. DE-AC02-05CH11231
H_Nishimura@lbl.gov

Based on our experience with Tracy++ and Tracy#, Tracy.Lite also has two implementations; Tracy.Lite++ in C++ and Tracy.Lite# in C#. Tracy.Lite++ is for the use on the HPC clusters and also on modern many-code CPUs. Tracy.Lite# is for the development efficiency, flexibility and the use from Python. By developing these two versions simltaneously, a better compatibility is established than the previous case.

Simplification

This is to make both design and implementation simple and concise to improve maintenance ability, development efficiency, execution speed and reliability.

The Element class models building blocks of the beam lines, such as drift spaces, various magnets, monitors and RF cavities. Tracy++ and Tracy# have the root class Element and its descendants which form a tree. This hierarchy has been removed and the single Element class supports all the Elements by distinguishing the types of building blocks by using the object attributes. This may look like a degeneration as a general Object-Oriented Programming practise, however, it makes external call from other programming languages, such as Python, much more transparent and feasible.

The lattice definition was done by using operation overloading effectively. As Tracy++ required complex memory management for it, Tracy.Lite++ uses generic containers as in Tracy# and Tracy.Lite#.

These effects are reflected in the library sizes as shown below.

Table 1: The Library Code Length in Lines

Library	Language	Core	ALS
Tracy++	C++	35K	6K
Tracy#	C#	22K	11K
Tracy.Lite++	C++	6K	2K
Tracy.Lite#	C#	5K	5K

Core is the body of the library not dedicated to ALS.
ALS is the model of various ALS lattice configurations.

Execution Speed

The speed up comes in the two directions; one is in a single thread mode by removing the hot spots where CPU time is wasted by using CPU profiling, and also simplifying the algorithms that are physically redundant in the ideal lattice design phase. The second is by multi-threading; Tracy.Lite++ uses OpenMP[9] and Tracy.Lite# uses Parallel.For[10] that is similar to OpenMP and a part of the .NET Framework.

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPWI020

The result bench marking largely depends on a routine and a type of the CPU therefore we do it with 5 different CPUs on the Windows OS.

Single-Treaded Case. Table 2 is a case with calculating the dynamic aperture of the original ALS lattice by using the transfer matrices in the 5-dim space for 400 turns on the 1 mm mesh points over +/- 40 mm in X and 0 to 20 mm in Y. All the calculation were done by using only one CPU core. There are cases Tracy.Lite# runs slower than Tracy# that uses tricks for faster speed by sacrificing the thread-safety.

Table 2: Execution Time on one CPU core [sec]

CPU	++	#	Lite++	Lite#
A	2.01	2.88	0.89	2.33
B	4.06	3.76	1.50	3.09
C	1.75	2.45	1.84	3.20
D	4.59	5.28	1.99	5.62
E	1.44	1.56	0.55	1.54

++ means Tracy++, # for Tracy#, Lite for Tracy.Lite.
A: Intel Xeon E5440 2.83 GHz Dual, 8 cores
B: AMS Opetron 6179 2.3 GHz, 8 cores, Hyper-V client
C: Intel Xeon E5-2670 2.6 GHz Dual, 32 cores
D: AMS Athlon 64 X2 3600+ 1.9 GHs, 2 cores
E: Intel Core i5-4440S 2.8 GHz, 4 cores

Multi-Threaded Case. The speed of Tracy.Lite in single and multi-threaded case are compared in Table 3. This is for the same dynamic aperture calculation but with the 2nd-order symplectic integrators in the 5-dim space with 16 segments per quad and bend. A notable observation is that there are cases of Parallel.For in C# showing better performance than OpenMP.

Table 3: Execution Time with/without Threading [sec]

CPU	++1	++N	#1	#N
A	8.50	1.75	9.70	2.33
B	12.13	2.84	16.82	4.92
C	8.07	0.78	8.52	0.73
D	14.71	12.12	18.50	10.63
E	6.22	3.02	8.08	3.74

++1: Tracy.Lite++ single-threaded
++N: Tracy.Lte++ nulti-threaded by OpenMP
#1: Tracy.Lite# single-threaded
#N: Tracy.Lite# multi-threaded by Parallel.For
A~E: same as in Table 2.

Multi-Threaded on Intel Xeon Phi. There are modern CPUs that come with many cores. We have just started using a PC system with two Intel Xeon Phi boards[11]. The boards provide total ~120 cores, or ~480 hyper-threads.

Compared with our previous experience of using GPU for the same type of calculation[12], Phi shows much better compatibility than GPU with the C++ code on the host PC because Phi uses Intel-type cores. For local parallel processing, OpenMP is used with 240 hyper-threads on each Phi board and CPUs on the host, and OpenMPI among Phi boards and the host. As of this writing, our Phi system is being commissioned. Once this port is completed, it can be scaled out to the lab's HPC cluster Lawrencium with 8 or more PHI boards.

Accessibility from Python

The .NET languages share the same binary format. Therefore, IronPython[13], a Python compiler on the .NET, can use Tracy# and Tracy.Lite# routines directly to provide interactive environment.

A standard Python can also call external libraries written in a .NET language without explicitly creating wrappers but by loading a module called Python for .NET[14]. Therefore, a standard Python can access Tracy# and Tracy.Lite# routines transparently, too. This includes the use form the IPython Notebook[15] therefore they can be used with other Python modules, such as MatPlotLib[16].

On the other hand, C/C++ routines must be explicitly wrapped in advance for the use from a standard Python. In case of Tracy++, its wrapper creation was not trivial due to its complexity, which was much simplified with Tracy.Lite++.

Platform Portability

Tracy.Lite++ is compatible with Visual C++ and GNU C++ therefore portable on various operating systems (OS). Tracy.Lite# run on the .Net Framework including MONO[17] that coves the Mac OS and most of the major Linux distributions, in addition to the Windows OS. However, at this moment, the performance of Tract.Lite# on non-Windows platform was not tested.

Smooth Switching between 5- and 6-dim spaces

The lattice parameter fitting and tracking routines must use the same integrator to be consistent. The most of the fittings use the 4x5 matrix formalism. For example, to calculate Twiss functions, Tracy uses a routine GetTwiss that implements the formulas[18] based on the synchrotron integrals that use transfer matrices in the bending magnets, and not compatible with the symplectic integrators. Tracy.Lite calculates matrices dynamically by using the same symplectic integrators for the use in GetTwiss. Therefore, the primary fitting routines in the 4x5 matrix formalism also work with the symplectic integrators in 5-dim. This makes the switching of integrators in 5-dim smooth, and the number of kicks per magnet can be reduced to save the CPU time. This speeds up the energy aperture calculation that

MOGA in the C# Version

Tracy.Lite# uses the C# routines of the Evolutionally Optimization[19] for many parameter fittings of multi-objective optimizations.

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2015). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPWI020

Content from this work may be used under the terms of the CC BY 3.0 licence (© 2015). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI.

Flexible Parameter Scan

A brute force parameter scan complements the results given by MOGA that returns highly optimized solutions. By scanning knobs in the vicinity of the region found by MOGA, we can have the landscape of the parameter space intuitively.

The number of scan parameters may be over 10 but unlikely over 20 or 30, therefore the straightforward scan is reasonably achievable. However, to be practical, the range and the step size of each parameter must be carefully chosen and dynamically adjusted. In addition, the order of the scan parameters may have to be reconfigured at run-time for effective load balancing on multiple CPU cores.

Tracy.Lite# implements a for-loop as a class with methods using the recursive calls and the function delegates effectively. This is being ported to Tracy.Lite++.

CONCLUSION

A new version of the library, Tracy.Lite, brings much faster and more flexible modeling and simulation capabilities than the current Tracy++ and Tracy#. Local parallel processing by using multi-threading fits well in the library. Local parameter scan in the vicinity of the solution given by MOGA recovers landscape of the parameter space intuitively. Interactive uses from Python, especially from the IPython Notebook, enable design studies with context by keeping the code and its document together.

ACKNOWLEDGMENT

We applicate T. Kellogg, C. Lam, B. Li and Y. Qin for their technical discussion and support.

REFERENCES

[1] H. Nishimura, EPAC'88, Rome, Italy, p.803 (1989)
J. Bengtsson, E. Forest and H. Nishimura, "Tracy User Manual", unpublished, ALS, LBNL
[2] LBL PUB-5172 Rev. LBL, 1986
A. Jackson, PAC'93, Washington, D.C, USA, p.1432, (1993)
[3] H. Nishimura, PAC'01, Chicago, USA, p.3006, (2001)
[4] L. Yang et. al., Nucl. Instr. and Meth. A 609, 50-57 (2009)
[5] C. Sun, et. al., Phys. Rev. STAB 15, 054001 (2012).
[6] C. Steier, et. al., IPAC 2014, Dresden, Germany, p567 (2014)
[7] http://www.open-mpi.org/
[8] H. Nishimura, ICAP 09, San Francisco, USA, (2009) http://accelconf.web.cern.ch/AccelConf/ICAP2009/papers/thpsc035.pdf
[9] http://openmp.org/wp/
[10] http://parallelpatterns.codeplex.com/
[11] http://www.intel.com/content/www/us/en/processors/xeon/xeon-phi-detail.html
[12] H. Nishimura, et. al., PAC '11, New York, USA, p.1764 (2011).
[13] http://ironpython.net/
[14] http://sourceforge.net/projects/pythonnet/files/PythonNet%20CLR%204.0/
[15] http://ipython.org/notebook.html
[16] http://matplotlib.org/
[17] http://www.mono-project.com/
[18] R.H. Helm et al., PAC'73, San Francisco, USA, p.900 (1973).
[19] J. McCaffrey, http://visualstudiomagazine.com/articles/2014/02/01/ evolutionary-optimization-using-c.aspx