

Project Team Sibin Mohan, Salil Pant, Harini Ramaprasad

{ smohan, smpant, hramapr }@unity.ncsu.edu

Our Project:

- Installation of Tuning and Analysis Utilities (TAU), a portable profiling and tracing toolkit, on the i32/Linux platform.
- > Evaluation using ParBenCCh (from the ASCI suite of benchmarks).

Project Topic – TAU Solved issues:

- We configured TAU to work with OpenMP and OpenMP-MPI hybrid programs. In order to get it to work, we had to make the following changes to the makefile:
 - o Include the library libompstub.a
 - Include the library libgm.so
- We ran an example program that uses OpenMP with TAU. The program involves calculations to generate the Mandelbrot set.
 - The results obtained from pprof are shown below.

NODE 0;CONTEXT 0;THREAD 0:							
~ %тіme	Exclusive msec	Inclusive total msec	#Call #	Subrs	Inclusive	Name usec/call	
100.0 99.1 0.9	0.505 20,064 172	20,238 20,065 172	1 1 1	800 2	20238593 20065216 172817	main() int (int, char **) Parallel Region ppmwrite() void (char *, field,	
int, 0 0.0 0.0	const Color 0.573 0.038	таble&) 0.573 0.038	800 1	0 0	1 38	foo ColorTable::~ColorTable() void	
0.0 (int,	0.013 int)	0.013	1	0	13	ColorTable::ColorTable() void	
Smootl 0.0	hColorTable 0.002	2::SmoothCol 0.002	orTable() 1	void 0	(int, direc 2	tion, base, base, base) Color⊤able::numColors() int	
0.0	0.001	0.001	1	0	1	ColorTable::shades() int (void)	

• A snap shot of jRacy outputs for the same program is shown below.



We ran an example program to test TAU for hybrid OpenMP-MPI. The program used is one that solves 2nd Stommel Model of Ocean Circulation using a Five-Point stencil and Jacobi iteration.

• The results obtained from pprof for this program are shown below.

		(total).				
%Time	Exclusive msec	Inclusive total msec	#Call	#Subrs	Inclusive /usec	Name call
100.0	29	4,550	2	6050	2275313	main() int (int, char
85.9	537	3,909	2000	800000	1955	do iacobi() void (FLT
**, FL 74.1	T **, FLT 3,372	**, INT, INT, 3,372	INT, INT) 800000	0	4	OpenMP Parallel for
(do_ja 4.8	cobi) 41	218	2000	16000	109	do_transfer() void (FLT
**, IN	T, INT, IN	T, INT) 210	2	16	105126	NDT Toit()
3.4	154	154	8000	10	105128	MPI_INICO
2.1	97	97	2	8	48588	MPI_Finalize()
1.5	67	67	2000	0 0	34	MPI_Reduce()
0.3	10	22	8000	Š.	685	MPI_Send()
0.1	10	4	2	ŏ	2382	do force() void (INT.
INT, I	NT, INT)		-			
0.1	0 226	0 236	4	0	619	MPI_Comm_split()
0.0	0.064	0.064	6	ŏ	11	matrix() FLT ** (INT.
INT, I	NT, INT)	0.001	ů –			
0.0 INT, I	0.013 NT, INT)	0.013	2	0	6	bc() void (FLT **, INT,
0.0	0.012	0.012	8	0	2	MPI_Attr_get()
0.0	0.01	0.01	8	o o	1	MPI_Attr_put()
0.0	0.004	0.007	ŝ	ŏ	1	MPI_Keyval_free()
0.0	0.003	0.003	ĕ	ŏ	ō	MPI_Comm_rank()
FUNCTIO	N SUMMARY	(mean): Inclusive	#call	#subrs	Inclusive	Name
FUNCTIO	n SUMMARY xclusive msec	(mean): Inclusive total msec	#call	#Subrs	Inclusive usec/c	Name all
FUNCTIO %Time E 100.0	N SUMMARY xclusive msec 14	(mean): Inclusive total msec 2,275	#Call 1	#Subrs 3025	Inclusive usec/c 2275313	Name call main() int (int, char
FUNCTIO %Time E 100.0 **) 85.9	xclusive msec 14 268	(mean): Inclusive total msec 2,275 1,954	#Call 1	#Subrs 3025 400000	Inclusive usec/c 2275313 1955	Name call main() int (int, char do_jacobi() void (FLT
FUNCTIO %Time E 100.0 **) 85.9 **, FLT	xclusive msec 14	(mean): Inclusive total msec 2,275 1,954 *, INT, INT,	#Call 1 1000 INT, INT)	#Subrs 3025 400000	Inclusive usec/c 2275313 1955	Name call main() int (int, char do_jacobi() void (FLT
FUNCTIO "Time E 100.0 **) 85.9 **, FLT 74.1 (do_jac	xclusive msec 14 268 FLT * 1,686 obi)	(mean): Inclusive total msec 2,275 2,275 *, INT, 1,954 1,686	#Call 1 1000 INT, INT) 400000	#Subrs 3025 400000 0	Inclusive usec/c 2275313 1955 4	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for
FUNCTIO """ *"" *"" *"" *"" *"" *"" *"	N SUMMARY xclusive msec 14 268 **, FLT * 1,686 obi) 20 , INT, INT	(mean): Inclusive total msec 2,275 *, INT, I,954 *, INT, I,686 , INT)	#call 1 1000 INT, INT) 400000 1000	#Subrs 3025 400000 0 8000	Inclusive usec/c 2275313 1955 4 109	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT
FUNCTIO %Time E 100.0 **) 85.9 **, FLT 74.1 (do_jac 4.8 **, INT 4.6	N SUMMARY xclusive msec 14 268 **, 268 FLT * 1,686 obi) 20 , INT, INT 105	(mean): Inclusive total msec 2,275 *, INT, INT, 1,686 , INT) 109 105	#call 1 1000 INT, INT) 400000 1000	#subrs 3025 400000 0 8000 8	Inclusive usec/c 2275313 1955 4 109 105126	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init()
FUNCTIO %Time E 100.0 **) 85.9 **, FLT 74.1 (do_jac **, INT 4.6 3.4	N SUMMARY xclusive msec 14 **, FLT * 1,686 obi) 20 , INT, INT 105 77	(mean): Inclusive total msec 2,275 *, INT, 1,954 *, INT, 1,686 109 , INT) 105 77 77	#Call 1 1000 INT, INT) 400000 1000 40000	#Subrs 3025 400000 0 8000 8 0	Inclusive usec/c 2275313 1955 4 109 105126 19 405126	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv()
FUNCTIO %Time E 100.0 **) 85.9 **, FLT 74.1 (do_jac 4.8 **, INT 4.6 3.4 2.1 1.5	N SUMMARY xclusive msec 14 268 FLT * 1,686 obi) 20 , INT, INT 105 77 48 33	(mean): Inclusive total msec 2,275 *, INT, INT, 1,686 109 , INT) 105 77 48 33	#Call 1 1000 INT, INT) 400000 1000 1000 1000	#subrs 3025 400000 0 8000 8 0 4 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Finalize() MPI_Finalize()
FUNCTIO %Time E 100.0 **) 85.9 **, FLT 74.1 (do_jac 4.8 **, INT 4.6 3.4 2.1 1.5 0.5	xclusive msec 14 268 **, FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11	(mean): Inclusive total msec 2,275 2,275 *, INT, 1,954 *, INT, 1,686 109 , INT) 105 77 48 31 11	#Call 1 1000 INT, INT) 400000 1000 1 4000 1 1000 4000	#Subrs 3025 400000 0 8000 8 0 4 0 4 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 34 3	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Finalize() MPI_Reduce() MPI_Red()
FUNCTIO %Time E 	xclusive msec 14 268 **, FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5	(mean): Inclusive total msec 2,275 2,275 *, INT, 1,954 *, INT, 1,686 109 , INT) 105 77 48 33 11 5	#Call 1 1000 INT, INT) 400000 1000 1000 1000 4000 200 8	#Subrs 3025 400000 0 8000 8 0 4 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 4 5685	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Reduce() MPI_Red() MPI_Bcast() MPI_Bcast()
FUNCTIO %Time E 	xclusive msec 14 268 **, FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 2 7	(mean): Inclusive total msec 2,275 *, INT, 1,954 *, INT, 1,686 109 *, INT) 105 77 48 33 11 5 2	#Call 1 1000 INT, INT) 400000 1000 1000 1000 1000 8 1	#Subrs 3025 400000 0 8000 8 0 4 0 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 4 8588 34 3 3 685 2382	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Finalize() MPI_Reduce() MPI_Reduce() MPI_Scast() do_force() void (INT,
FUNCTIO """ """ """ """ """ """ """ "	xclusive msec 14 268 **, FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 2 T, INT) 1	(mean): Inclusive total msec 2,275 *, INT, 1,954 *, INT, 1,686 109 *, INT) 105 77 48 33 11 5 2 1	#Call 1 1000 INT, INT) 400000 1000 1000 1000 4000 8 1 2	#Subrs 3025 400000 0 8000 8 0 4 0 0 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 4 8588 34 3 685 2382 619	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Reduce() MPI_Reduce() MPI_Reduce() MPI_Reduce() MPI_Bcast() do_force() void (INT, MPI Comm split()
FUNCTIO """ """ """ """ """ """ """ "	xclusive msec 14 268 FLT * 1,686 obi) 20 , INT, INT, 105 77 48 33 11 5 2 T, INT) 1 0.118	(mean): Inclusive total msec 2,275 *, INT, 1,954 *, INT, 1,686 , INT) 109 , INT) 105 77 48 33 11 5 2 0.118	#Call 1 1000 INT, INT) 400000 1000 4000 1000 4000 8 1 2 2	#Subrs 3025 400000 0 8000 8 0 4 0 0 0 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 48588 34 3 685 2382 619 59	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Reduce() MPI_Reduce() MPI_Reduce() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_Wtime()
FUNCTIO %Time E 100.0 **) 85.9 **, FLT 74.1 (do_jac 4.8 **, INT 4.6 3.4 2.1 1.5 0.2 0.1 INT, IN 0.1 0.0 0.0	N SUMMARY xclusive msec 14 268 FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 2 T, INT) 0.118 0.032	(mean): Inclusive total msec 2,275 *, INT, INT, 1,686 *, INT) 109 105 77 48 33 11 5 2 0.118 0.032	#Call 1 1000 INT, INT) 400000 1000 4000 1000 4000 8 1 2 2 3	#Subrs 3025 400000 0 8000 8 0 4 0 0 0 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 685 2382 619 59 11	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Reduce() MPI_Reduce() MPI_Reduce() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_Wtime() matrix() FLT ** (INT,
FUNCTIO "Time E "Time E 100.0 **) **, FLT 74.1 (do_jac 4.8 **, INT 4.6 3.4 2.1 1.5 0.2 0.1 INT, IN 0.1 0.0 INT, IN 0.0	N SUMMARY xclusive msec 14 268 **. FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 2 T, INT) 0.118 0.032 T, INT) 0.035	(mean): Inclusive total msec 2,275 *, INT, INT, 1,686 *, INT) 109 , INT) 105 77 48 33 11 5 2 0.0118 0.025	#call 1 1000 INT, INT) 400000 1000 4000 1 1000 4000 8 1 2 2 3 1	#Subrs 3025 400000 0 8000 8 0 4 0 0 0 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 685 2382 619 59 11	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Reduce() MPI_Reduce() MPI_Reduce() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_Wtime() matrix() FLT ** (INT, bc() void (FLT ** INT,
FUNCTIO "Time E "Time E 100.0 **) **, FLT 74.1 (do_jac 4.8 **, INT 4.6 3.4 2.1 1.5 0.2 0.1 INT, IN 0.0 INT, IN 0.0 INT, IN 0.0 INT, IN	N SUMMARY xclusive msec 14 268 **. FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 2 T, INT) 0.118 0.032 T, INT) 0.0065 T, INT)	(mean): Inclusive total msec 2,275 *, INT, INT, 1,686 *, INT) 109 , INT) 105 77 48 33 11 0.5 2 0.0065	#call 1 1000 INT, INT) 400000 1000 4000 1000 4000 8 1 2 2 3 1	#Subrs 3025 400000 0 8000 8000 8 0 4 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 685 2382 619 59 11 6	Name call main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Init() MPI_Recv() MPI_Reduce() MPI_Reduce() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_Comm_split() MPI_Wtime() matrix() FLT ** (INT, bc() void (FLT **, INT,
FUNCTIO "Time E "Time E 100.0 **) **, FLT 74.1 (do_jac 4.8 **, INT 4.6 3.4 2.1 1.5 0.2 0.1 INT, IN 0.0 INT, IN 0.0 INT, IN 0.0 INT, IN 0.0	N SUMMARY xclusive msec 14 268 **. FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 7, INT) 0.118 0.032 T, INT) 0.0065 T, INT) 0.006	(mean): Inclusive total msec 2,275 *, INT, I,954 *, INT, I,686 , INT) 109 , INT) 105 77 48 33 11 0.118 0.032 0.0065 0.006	#call 1 1000 INT, INT) 400000 1000 4000 1000 4000 8 1 2 2 3 1 4 4	#Subrs 3025 400000 0 8000 8000 8 0 4 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 84 3 685 2382 619 59 11 1 6 2	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Recv() MPI_Recv() MPI_Reduce() MPI_Send() MPI_Send() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_Comm_split() MPI_Wtime() matrix() FLT ** (INT, bc() void (FLT **, INT, MPI_Attr_get()
FUNCTIO """ """ """ """ """ """ """ "	N SUMMARY xclusive msec 14 268 **. FLT * 1,686 obi) 20 , INT, INT 105 77 48 33 11 5 2 T, INT) 0.118 0.032 T, INT) 0.0065 T, INT) 0.0065 T, INT) 0.0065	(mean): Inclusive total msec 2,275 *, INT, I,954 *, INT, I,686 , INT) 109 , INT) 105 77 48 33 11 5 2 0.032 0.0065 0.006 0.005	#call 1 1000 INT, INT) 400000 1000 1000 4000 4000 8 1 2 2 3 1 4 4 4 4	#subrs 3025 400000 0 8000 8 0 4 0 0 0 0 0 0 0 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 685 2382 619 59 11 6 2 1	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Recv() MPI_Recv() MPI_Reduce() MPI_Send() MPI_Send() MPI_Send() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_wtime() matrix() FLT ** (INT, bc() void (FLT **, INT, MPI_Attr_get() MPI_Attr_put()
FUNCTIO """ """ """ """ """ """ """ "	N SUMMARY xclusive msec 14 268 **. FLT * 1,686 obi) 20 , INT, INT 0,118 0.032 T, INT) 0.0065 T, INT) 0.0065 T, INT) 0.0065 0.0035 0.002	(mean): Inclusive total msec 2,275 *, INT, INT, 1,686 *, INT) 109 , INT) 105 77 48 33 11 5 2 1 0.032 0.0065 0.006 0.005 0.005 0.005 0.005	#call 1 1000 INT, INT) 400000 1000 1000 4000 4000 8 1 2 2 3 1 4 4 4 4 3	#subrs 3025 400000 0 8000 8000 8000 8000 8000 0 0 0	Inclusive usec/c 2275313 1955 4 109 105126 19 48588 34 3 685 2382 619 59 11 6 2 2382 619 59 11 6 2 1 1	Name all main() int (int, char do_jacobi() void (FLT OpenMP Parallel for do_transfer() void (FLT MPI_Recv() MPI_Recv() MPI_Finalize() MPI_Send() MPI_Send() MPI_Scast() do_force() void (INT, MPI_Comm_split() MPI_Wtime() matrix() FLT ** (INT, bc() void (FLT **, INT, MPI_Attr_get() MPI_Keyval_free() MPI_Keyval_free() MPI_Keyval_free() MPI_Keyval_free()

• The results obtained from jracy for this program are in the next figure. We see that the maximum time is spent in the omp parallel for sections while the next largest is for the calculation of the jacobians.



Project Topic – ParBenCCh Solved issues:

- We installed and completely configured ParBenCCh. This required the following changes to be made to the makefile:
 - Include the library libompstub.a
 - Include the library libguide.so

- Once ParBenCCh was installed, we had to configure the various tests in the suite. The tests in the ParBenCCh suite are briefly described below (information obtained from the README on the website <u>http://www.llnl.gov/asci/purple/benchmarks/limited/parbencch/p</u> <u>arbencch.readme.html</u>):
 - The Haney Test

This test compares the performance of matrix-vector operations in the following settings:

- ▲ *Real matrix multiplication*
- ▲ Complex matrix multiplication
- Real vector operations to test the cost of overloaded operators for operations on arrays
- o The Stepanov Test

This test measures the compiler support and performance of expression templates – a C++ template mechanism intended to achieve FORTRAN-like performance.

• The OpenMP Test

This is a test of OpenMP-style parallel direct and indirect addressing. In this test, a one-dimensional array of doubles whose size is close to the maximum heap size is allocated. The following operations are then performed on this array:

- ▲ Linear read
- ▲ Linear read-write
- ▲ Random read
- ▲ Random read-write
- Tests for Indirect Addressing

The tests in this directory exercise parallel indirect addressing using MPI-based parallelism. There are three related tests that are performed here.

Project Topic – running ParBenCCh tests with TAU Solved issues:

- ➤ We ran the tests in the ParBenCCh suite using TAU. In order to be able to run the tests with TAU, the source code was instrumented according to instructions provided in the TAU installation guide.
- ➤ We first had to make the following changes to the makefile for the tests.
 - Include the TAU makefile stub (found in TAU_ROOT /i386_linux/lib directory) in the makefile of each test.
 - Add TAU_INCLUDE and TAU_DEFS in CXXFLAGS, CXXINCLUDE and CFLAGS.
 - Add –ltau, -lpthread and –lstdc++ in LIBS
 - Include the library /opt/papi/lib/libpapi.a
 - Include the TAU library TAU_ROOT/i386_linux/lib/libtaumpi-pthread-papi-pdt-openmp.a
- The next step was the actual instrumentation of the source code of each test. The process is as follows:
 - Parse the source file using cxxparse <source_file> to generate a .pdb file.
 - Instrument the source file using tau_instrumentor <pdb file> <source file>
- > Make and run the instrumented test programs.
- View the performance results as text using pprof and a graphical output using jRacy.
- The results obtained from these tests are shown below.
 The Haney Test

The results for the Haney Benchmark on pprof::

NODE 0;CONTEXT 0;THREAD 0: --------%Time Exclusive Inclusive #Call #Subrs Inclusive Name msec total msec use usec/call 100.0 1 4:30.521 36.7 1:39.394 1:39.394 100.0 8 270521253 int main(int, char **) 1 28 0 3549819 void HaneyBench::testRMatMul(int, double *) 12.6 33,983 33,983 28 336 1213684 void HaneyBench::testMatMul(int, double *) 0.0 0.373 0.373 1 84 373 void HaneyBench::writeLabels(std::ostream &) 0.0 0.001 0.06 1 0.0 0.034 0.034 84 28 60 void_HaneyBench::finalize() 0 0 ComplexArray3 &ComplexArray3::ComplexArray3(Integer, Integer, Integer, Boolean)
0.0
0.029
0.029
84
0
ComplexArray4
 0.0
 0.002
 0.002
 1

 &HaneyBench::HaneyBench(int, int, int)
 0.001
 1
 28 1 void HaneyBench::initialize(int, char **) 84 0 void 0.0 0 0 1 HaneyBench::runBenchmark() 0.0 0 1 0 0 void 0 HaneyBench::setArraySize(int) 0.0 0 Ó 28 0 0 void HaneyBench::testVecOps(int, double *) 0.0 0 1 0 0 void HaneyBench::writeToFile(std::ostream &) 1 0.Ó 0 0 void 0 HaneyBench::~HaneyBench()

The above results show that the maximum number of calls were made to the constructors of the ComplexArray3 and ComplexArray4 classes. The most time was spent in the HaneyBench::testRMatMul() and HaneyBench::testMatMul() methods of the HaneyBench class.

\circ jRacy outputs for the program are shown below.

Bacy Main Window: pprof.dat/Haney/ParBenCCh-1.1.2/ParBenCCh/CSC_591C/smohan/home/ ile Options Windows Help	Function Data Window: pprof.dat/Haney/ParBenCCh-1.1.2/ParBenCCh/CSC_591C/smohan/
Mean	COUNTER NAME: void Haney8ench::testMatMul(int, double *) 12.49% 12.49% n,c,t 0,0,0 v
Function Data Window: pprof.dat/Haney/ParBenCCh-1.1.2/ParBenCCh/CSC_SGIC/smohan/home/ Ile Options Windows Help COUNTER NAME: Default FUNCTION NAME: void Haney@ench::testVecOps(int, double *) 50.7% mean 50.7% n, c,t 0,0,0	Function Data Windows pprof.det/Haney/ParBentCh-L1.2/ParBentCh/LSC_S91C/ File Options Windows Help COUNTER NAME: Default FUNCTION NAME: void Haney8ench::testRMatMul(int, double*) 36.81% mean 36.81% n.c,t 0,0,0

- The Stepanov Test
 - Instrumentation for this test was causing segmentation faults. The Non-instrumented program was executing normally though.
- The Indirect Addressing Tests

The results obtained from TAU for this test are as follows :

1	🗶 jRacy I	Main W	indow: pprol	i.dat/IndirectAddressing/ParBenCCh-1.1.2/ParBenCCh/CSC_591C/smohan/home/	- D ×
2	File Op	tions	Windows	Help	
	COUNTE	R NAM Mean	E: Default		
	n, c, t	0,0,0			•

This shows the percentage of time spent in various functions are shown above. The functions and the color keys are :



the MPI_Irecv_wrap() and MPI_send_wrap() are wrappers in the benchmark for calling MPI functions.

Various statistics obtained for this program are (number of calls, number of subroutine calls, per-call-value and time spent inclusive of profiling, in milliseconds) :



[where n – node, c – context, t – thread]

These results are for the testoo() function. It shows that this particular function was called 13 times, called other subroutines 26 times, took 308.0 microseconds for each invocation, and also that it took totally 4.01 milliseconds to execute all of it's invocations.

We also see that this particular function was 3rd in the most time consuming functions in this particular test (as seen from the very first image).

The pprof output for the same : [smohan@os07 IndirectAddressing]\$ pprof Reading Profile files in profile.* NODE 0; CONTEXT 0; THREAD 0: _ _ _ _ Exclusive Inclusive #Call #Subrs Inclusive Name msec total msec usec/call %Time 100.0 10 10 13 26 844 void test04(long DistMappingType, int, int, long, DistMappingType, int, int, const std::vector<long, 13 844 void test04(long, std::allocator<long>> &) 5 54.4 13 459 void test25(long, 0 DistMappingType, int, int, long, DistMappingType, int, int, const std::vector<long, std::allocator<long>> &) 36.5 13 26 308 void test00(long, 4 3 DistMappingType, int, int, long, DistMappingType, int, int, const std::vector<long, std::allocator<long>> &) 0.044 26 0 2 int 0.4 0.044 MPI_Isend_wrap(const type_tag<long> &, void *, int, int, int, MPI_Comm, MPI_Request 0.2 0.026 0.026 26 0 1 int MPI_Irecv_wrap(const type_tag<long> &, void *, int, int, int, MPI_Comm, MPI_Request 0.2 0.026 0.026

Problems encountered and Changes required in TAU/ParBenCCh

We faced the following problems and we have listed what we believe should be improvements to the Profiler/Benchmark.

- Instrumentation of programs This was not an easy task, and involved a lot of intricate changes to the Makefiles, as well as source code, on our part.
- The libraries and paths were not being set and had to be often set manually, with the full path.
- Many coding bugs still exist in the Benchmark which made not only instrumentation difficult, but also just running the benchmark as is.
- The sample programs provided along with TAU also had problems in their source code and Makefiles – for eg. : the OpenMP sample provided wouldn't compile as is – there were problems in the Makefile which had to be corrected, even before the instrumentation.
- The automatic source-code instrumentor, tau_instrument, would often produce incorrect C++/C output files and put instrumentation code in the wrong places which would

- In certain cases, when comprehensive instrumentation was requested (for eg. : instrumentation of all source files in the Haney Test), Segmentation faults would occur, and hence the program/profiling would crash.
- In certain cases, (for Eg. : Stepanov test) any form of automatic instrumentation would cause Segmentation faults. Hence we were unable to obtain profile data for the same.