**TAU - Tuning and Analysis Utilities**

**Introduction**

For definitive information, refer to [https://www.cs.uoregon.edu/research/tau/docs/usersguide/index.html](https://www.cs.uoregon.edu/research/tau/docs/usersguide/index.html)

**How to build**

**Requirements**

1. PDT
2. PAPI

**Tips for BlueGene/L**

**PAPI**

3.7.2 was successfully built with a few changes

1. Modify Makefile.linux-bgl.

   **(Expand for details) Makefile.linux-bgl**
   
   PREFIX= /hpc/projects/packages/local.bgl/pkg/papi/3.7.2
   CFLAGS = -D_BGL

2. Make a change to ctests/bgl_tests/hummer-loops.c to correct the following compile error. (Expand to see the details)

   "hummer-loops.c", line 25.21: 1506-025 (S) Operand must be a modifiable lvalue.

   **(Expand for details) ctests/bgl_tests/hummer-loops.c**
   
   int fmaddvSimple(int n ,double *a,double *b,double *c, double *t) {
   /* This function call generates N FMADD operations
   * 3N FP LOADS and N FP STORES (when compiled with optimization)
   * t = a * b + c
   */
   int i;
   for(i=n;i;i--) {
     a++;
     b++;
     c++;
     t++;
     asm volatile("fmadd %0,%1,%2,%3" : "=f"(*t) : "f"(*a), "f"(*b),
     "f"(*c));
   }
   return n;
#export CC=mpixlc
#.configure --with-OS=bgl

Building TAU

./configure -arch=bgl -mpi -pdt=/hpc/projects/packages/local.bgl/pkg/pdt/pdtoolkit-3.10 -pdt_c++=mpixlcxx -PROFILEMEMORY -PROFILEHEADROOM -BGLTIMERS -papi=/hpc/projects/packages/local.bgl/pkg/papi/3.7.2

without PAPI

./configure -arch=bgl -mpi -pdt=/hpc/projects/packages/local.bgl/pkg/pdt/pdtoolkit-3.10 -pdt_c++=mpixlcxx -PROFILEMEMORY -PROFILEHEADROOM -BGLTIMERS -TRACE -MPITRACE -PROFILECALLPATH -cc=/bgl/BlueLight/ppcfloor/bglsys/bin/mpixlc

Tips for Linux PC

Building TAU

./configure -mpi -pdt=$PDT/../ -mpi -papi=/usr/local -TRACE -MPITRACE -PROFILECALLPATH -PROFILECOMMUNICATORS -PROFILEMEMORY -PROFILEHEADROOM

Using TAU - Simplest way

Uninstrumented code

%mpirun -np 8 ./a.out

With TAU
%mpirun -np 8 tau_exec ./a.out

Then run paraprof

%paraprof

To have an access to all other features, it is recommended to instrument the code explained below. If you wish to profile on BlueGene, the program executed by mpirun must be a binary- tau_exec is a script and mpirun will reject to run it.

**Using TAU - Advanced way**

**Before using TAU - Set environment variables**

**BlueGene/L**

```
export TAU=/hpc/projects/packages/local.bgl/pkg/tau/version
export PDT=/hpc/projects/packages/local.bgl/pkg/pdt/pdtoolkit-3.10/bgl
export TAU_MAKEFILE=$TAU/lib/Makefile.tau-mpi-pdt
```

**Linux PC**

```
export TAU=$HOME/pkg/tau-2.24/x86_64
export PDT=$HOME/pkg/pdtoolkit-3.20/x86_64
export TAU_MAKEFILE=$TAU/lib/Makefile.tau-memory-mpi-pdt
```

Choose the most appropriate TAU_MAKEFILE available in $TAU/lib/

**Runtime environment variables**

Set runtime environment variables depending on your requirement. For example, if you wish to generate communication matrix,

```
export TAU_COMM_MATRIX=1
```

All available runtime environment variables (version 2.2.4)

- Show all options

  Unknown macro: `viewpdf`

  and you also need to set TAU_OPTIONS
export TAU_OPTIONS='--optVerbose --optMemDbg'

A list of options for the TAU compiler scripts can be found by typing tau_compiler.sh

Tau 2.24 supports following options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-optVerbose</td>
<td>Turn on verbose debugging message</td>
</tr>
<tr>
<td>-optMemDbg</td>
<td>Enable TAU's runtime memory debugger</td>
</tr>
<tr>
<td>-optDetectMemoryLeaks</td>
<td>Synonym for -optMemDbg</td>
</tr>
<tr>
<td>-optPdtDir=&quot;&quot;</td>
<td>PDT architecture directory. Typically $(PDTDIR)</td>
</tr>
<tr>
<td>-optPdtF950pts=&quot;&quot;</td>
<td>Options for Fortran parser in PDT (f95parse)</td>
</tr>
<tr>
<td>-optPdtF95Reset=&quot;&quot;</td>
<td>Reset options to the Fortran parser to the given list</td>
</tr>
<tr>
<td>-optPdtC0pts=&quot;&quot;</td>
<td>Options for C parser in PDT (cparse). Typically $(MPI_INCLUDE) $(MPI_DEFS)</td>
</tr>
<tr>
<td>-optPdtCReset=&quot;&quot;</td>
<td>Reset options to the C parser to the given list</td>
</tr>
<tr>
<td>-optPdtCxx0pts=&quot;&quot;</td>
<td>Options for C++ parser in PDT (cxxparse). Typically $(MPI_INCLUDE) $(MPI_DEFS)</td>
</tr>
<tr>
<td>-optPdtCxxReset=&quot;&quot;</td>
<td>Reset options to the C++ parser to the given list</td>
</tr>
<tr>
<td>-optPdtF90Parser=&quot;&quot;</td>
<td>Specify a different Fortran parser. For e.g., f90parse instead of f95parse</td>
</tr>
<tr>
<td>-optPdtGnuFortranParser</td>
<td>Specify the GNU gfortran PDT parser gparse instead of f95parse</td>
</tr>
<tr>
<td>-optPdtCleanscapeParser</td>
<td>Specify the Cleanscape Fortran parser</td>
</tr>
<tr>
<td>-optPdtUser=&quot;&quot;</td>
<td>Optional arguments for parsing source code</td>
</tr>
<tr>
<td>-optTauInstr=&quot;&quot;</td>
<td>Specify location of tau_instrumentor. Typically $(TAUROOT)/$(CONFIG_ARCH)/bin/tau_instrumentor</td>
</tr>
<tr>
<td>-optPreProcess</td>
<td>Preprocess the source code before parsing. Uses /usr/bin/cpp -P by default.</td>
</tr>
<tr>
<td>-optContinueBeforeOMP</td>
<td>Insert a CONTINUE statement before !$OMP directives.</td>
</tr>
<tr>
<td>-optCPP=&quot;&quot;</td>
<td>Specify an alternative preprocessor and pre-process the sources.</td>
</tr>
<tr>
<td>-optCPP0pts=&quot;&quot;</td>
<td>Specify additional options to the C pre-processor.</td>
</tr>
<tr>
<td>-optCPPReset=&quot;&quot;</td>
<td>Reset C preprocessor options to the specified list.</td>
</tr>
<tr>
<td>-optTauSelectFile=&quot;&quot;</td>
<td>Specify selective instrumentation file for tau_instrumentor.</td>
</tr>
<tr>
<td>-optTauWrapFile=&quot;&quot;</td>
<td>Specify path to the link_options.tau file generated by tau_wrap</td>
</tr>
<tr>
<td>-optTrackIO</td>
<td>Specify wrapping of POSIX I/O calls at link time.</td>
</tr>
<tr>
<td>-optTrackUPCR</td>
<td>Specify wrapping of UPC runtime calls at link time.</td>
</tr>
<tr>
<td>-optTrackDMAPP</td>
<td>Specify wrapping of Cray DMAPP library calls at link time.</td>
</tr>
</tbody>
</table>
-optTrackPthread Specify wrapping of Pthread library calls at link time.
-optTrackGOMP Specify wrapping of GOMP library calls at link time. (default)
-optNoTrackGOMP Disable wrapping of GOMP library calls at link time.
-optTrackMPCThread Specify wrapping of MPC Thread library calls at link time.
-optsWrappersDir Specify the location of the link wrappers directory.
-optsPDBFile Specify PDB file for tau_instrumentor. Skips parsing stage.
-optsTau Specify options for tau_instrumentor
-optsCompile Options passed to the compiler by the user.
-optsTauDefs Options passed to the compiler by TAU. Typically $(TAU_DEFS)
-optsTauIncludes Options passed to the compiler by TAU. Typically $(TAU_MPI_INCLUDE) $(TAU_INCLUDE)
-optsIncludeMemory Flags for replacement of malloc/free. Typically -I$(TAU_DIR)/include/TauMemory
-optsReset Reset options to the compiler to the given list
-optsLinking Options passed to the linker. Typically $(TAU_MPI_FLIBS) $(TAU_LIBS) $(TAU_CXXLIBS)
-optsLinkReset Reset options to the linker to the given list
-optsLinkPreserveLib Libraries which TAU should preserve the order of on the link line see "Moving these libraries to the end of the link line: ". Default: none.
-optsTauCC Specifies the C compiler used by TAU
-optsUseCXXForC Specifies the use of a C++ compiler for compiling C code
-optsUseReturnFix Specifies the use of a bug fix with ROSE parser using EDG v3.x
-optsOpariTool Specifies the location of the Opari tool
-optsLinkOnly Disable instrumentation during compilation, do link in the TAU libs
-optsOpariDir Specifies the location of the Opari directory
-optsOpariOpts Specifies optional arguments to the Opari tool
-optsOpariNoInit Do not initlize the POMP2 regions.
-optsOpariLibs Specifies the libraries that have POMP2 regions. (Overrides optOpariNoInit
-optsOpariReset Resets options passed to the Opari tool
-optsOpari2Tool Specifies the location of the Opari tool
-optsOpari2ConfigTool Specifies the location of the Opari tool
-optsOpari2Opts Specifies optional arguments to the Opari tool
-optsOpari2Reset Resets options passed to the Opari tool
-optsOpari2Dir Specifies the location of the Opari tool
directory
-optsNoMpi Removes -l*mpi* libraries during linking
-optsMpi Does not remove -l*mpi* libraries during linking
Compile your code with TAU compiler

You will be using TAU’s compiler wrapper (tau_cc.sh, tau_cxx.sh, tau_f77.sh, tau_f90.sh). This will add instrumentation to your original codes, and compile them.

If you have a single source, just use one of these compiler wrappers.

```
%tau_cc.sh hello.c -o hello
```

If you have a Makefile, edit CC (or CPP, CXX etc) and use one of compiler wrappers or alternatively,

```
%make CC=tau_cc.sh
```

Run

Run your program normally. It will be a lot slower than usual. This will generate a number of profile.* files which keep profile info.

Analyzing the result

paraprof is GUI based and provides rich visualisation options. Just enter paraprof in the directory where your profile.* files are located. It will automatically load them and display the output.
Alternatively, pprof is terminal-based, but provides more information (memory usage, memory leak etc).

### pprof: Memory leak detection

...  
USER EVENTS Profile : NODE 0, CONTEXT 0, THREAD 0  
---------------------------------------------------------------  
--------  
NumSamples MaxValue MinValue MeanValue Std. Dev. Event Name  
---------------------------------------------------------------  
--------  
   2   52    48     50     2 MEMORY LEAK!  
malloc size <file=simple.inst.cpp, line=18> : int g(int) => int bar(int)  
     1   80    80     80     0 free size  
     1   80    80     80     0 free size  
     1  180   180   180     0 free size  
     3   80    48     60   14.24 malloc size  
     3   80    48     60   14.24 malloc size  
     1  180   180   180     0 malloc size  
     1  180   180   180     0 malloc size  
acion that the first row show the two Memory leaks along with the callpath tracing where the unallocated memory was requested.