Dynamic Performance Callstack Sampling: Merging TAU and DAQV-II

Sameer Shende, Steven T. Hackstadt and Allen D. Malony

Computational Science Institute,
Department of Computer and Information Science
University of Oregon, Eugene, OR 97403, USA
{sameer, hacks, malony}@cs.uoregon.edu

http://www.cs.uoregon.edu/research/paracomp/proj/tau
http://www.cs.uoregon.edu/research/paracomp/proj/asci/daqv/

Tuning and Analysis Utilities

Callstack Monitoring

- Callstack: The current execution location and the sequence of procedure calls that led to that point
- Profiling, Tracing - post-mortem means of performance analysis
- Callstack monitoring - monitors the currently executing function and performance metrics associated with the functions on the callstack
- Parallelism - creates multi-dimensional, distributed “global” profile data

Implementation of Dynamic Callstack Sampling

We need ...

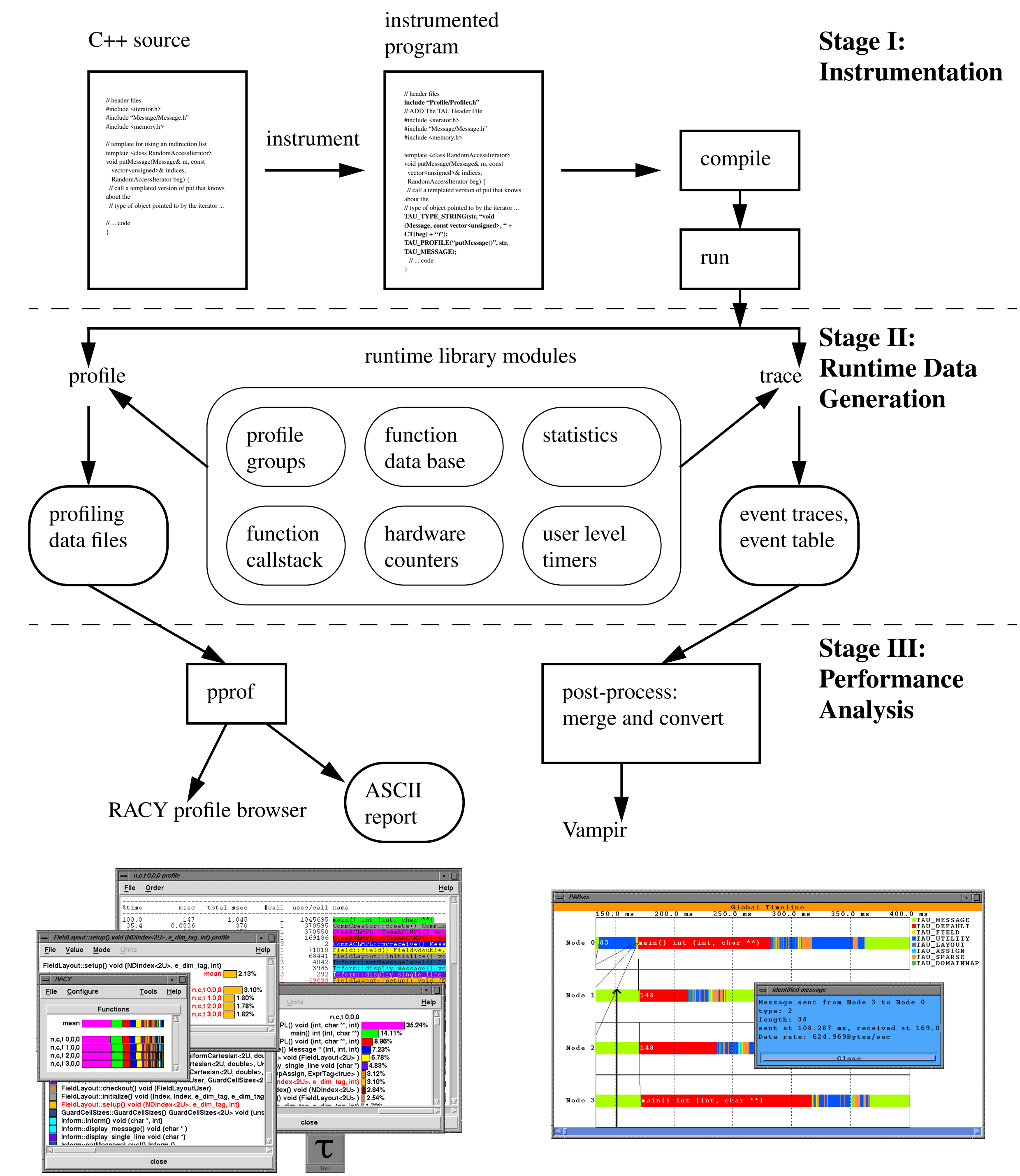
- Portable profiling framework with access to profiling data and callstack (TAU)
- Access and transport mechanism for multi-dimensional callstack data (DAQV-II)
- Visualization tools for displaying the callstack monitor

TAU: Tuning and Analysis Utilities

Tau is...

- An extensible tool framework supporting tool interactions with the program, the user, and each other
- A graphical, program development environment with several distinct and unique, but completely integrated, tools
- A performance analysis environment facilitating *static* and *dynamic* interactions with programs

TAU Portable Profiling and Tracing



TAU Portable Profiling

- Portable profiling and tracing facility for ISO/ANSI C++
  - multiple compilers, operating systems and platforms
  - templates
  - user-defined and statement level timers
- Modular and configurable profiling infrastructure
- Selective profiling - limit overhead to groups of profiled functions selected at runtime
- Lightweight profiling facility, minimal intrusion
- Timing-based instrumentation (not sampling) and support for hardware performance counters
- User instrumentation of source code via profiling API

Instrumentation in TAU

- Instrumentation is inserted in the source code either manually or by a preprocessor.
- Function (block) entry and exit points are used to record the exact time spent in the instrumented function.
- Statistics are maintained in the instrumented program and small profile files are generated at the end of execution.
- Supported for C++ and its parallel derivatives (pC++, HPC++)
- Compiler support is needed for automatic instrumentation (Sage++, EDG).
- Suitable for C++ libraries, applications where source code is available for instrumentation.

TAU Profiling API

- **TAU_PROFILE(func_name, type_string, profile_group);**

Inserted in each function that is to be profiled. e.g.,

```
int main(int argc, char **argv){
   TAU_PROFILE("main()", "int (int, char **)",
         TAU_DEFAULT);
```

- **TAU_TYPE_STRING(string_varname, string);**
- **TAU_PROFILE(func_name, string_varname, profile_group);**
- **CT(object);**

For profiling template member functions.

- **TAU_PROFILE_TIMER(var, name, type, group);**
- **TAU_PROFILE_START(var);**
- **TAU_PROFILE_STOP(var);**

To start and stop a timer. To time one or more statements in the code.

```
TAU_PROFILE_TIMER(timer1,"main-loop1", "int (int, char
    **)", TAU_USER);
        ...
        TAU_PROFILE_START(timer1);
         for(i=0; i < N; i++) {// loop1 profiled using
             timer1 var }
        TAU_PROFILE_STOP(timer1);
```

- **TAU_MONITOR();**

  For callstack monitoring

- **TAU_PROFILE_STMT(stmt);**

To declare a variable that is used only during profiling.

- **TAU_PROFILE_INIT(argc, argv);**
- **TAU_PROFILE_SET_NODE(myNode);**

To initialize profile groups and to set the current node id for each context

- **TAU_PROFILE_EXIT(const char *message);**

To abort the program, and dump profiling data to disk.

- **TAU_TRACE_SENDMSG(type, destination, length);**
- **TAU_TRACE_RECVMSG(type, source, length);**

Tuning and Analysis Utilities

Selective Profiling

Table 1: Selecting profile groups on the command line at runtime

ProfileGroup	Description	Runtime Identifier	Example
TAU_DEFAULT	All profiling groups enabled		--profile
TAU_MESSAGE	Message class includes MPI, PVM, ACLMPL	'm'	--profile M
TAU_PETE	Portable Expression Template Engine	'p'	--profile pete+message
TAU_VIZ	ACLVIZ	'v'	--profile Pete+viz
TAU_ASSIGN	Assign expression	'a'	--profile a+m
TAU_IO	IO functions	'i'	--profile io
TAU_FIELD	Field functions	'f'	--profile field+viz+assign
TAU_LAYOUT	Field Layout	'l'	--profile layout
TAU_SPARSE	Sparse Index	's'	--profile sparse+assign
TAU_DOMAINMAP	Domain Map	'd'	--profile i+d+viz
TAU_UTILITY	Utility	'ut'	--profile utils+io
TAU_USER	User	'u'	--profile user+region
TAU_USER1	User1	'1'	--profile 1+d
TAU_USER2	User2	'2'	--profile 2+d
TAU_USER3	User3	'3'	--profile 3+d
TAU_USER4	User4	'4'	--profile 4+d

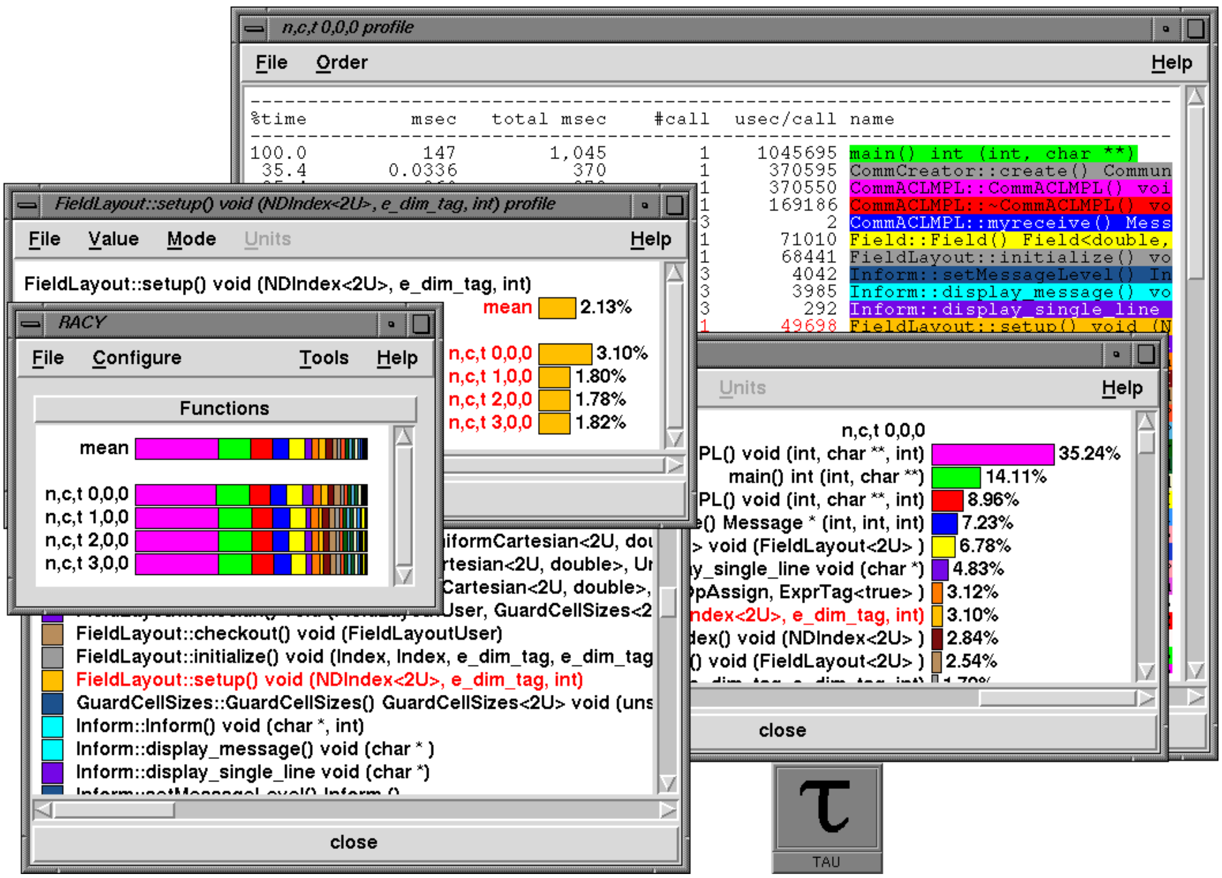
% mpirun -np 4 app --profile io+field+message

SGI R10000 Performance Counters

Table 1: Values of T5_EVENT0 environment variable

	Quantity Profiled		Quantity Profiled
0	Cycles	16	Graduated cycles
1	Issued instructions	17	Graduated instructions
2	Issued loads	18	Graduated loads
3	Issued stores	19	Graduated stores
4	Issued store conditionals	20	Graduated store conditionals
5	Failed store conditionals	21	Graduated floating point instructions
6	Decoded branches	22	Quadwords written back from primary data cache
7	Quadwords written back from scache	23	TLB misses
8	Correctable scache data array ECC errors	24	Mispredicted branches
9	Primary instruction cache misses	25	Primary data cache misses
10	Secondary instruction cache misses	26	Secondary data cache misses
11	Instruction misprediction from scache way prediction table	27	Data misprediction from scache way prediction table
12	External interventions	28	External intervention hits in scache
13	External invalidations	29	External invalidation hits in scache
14	Virtual coherency conditions	30	Store/prefetch exclusive to clean block in scache
15	Graduated instructions	31	Store/prefetch exclusive to shared block in scache

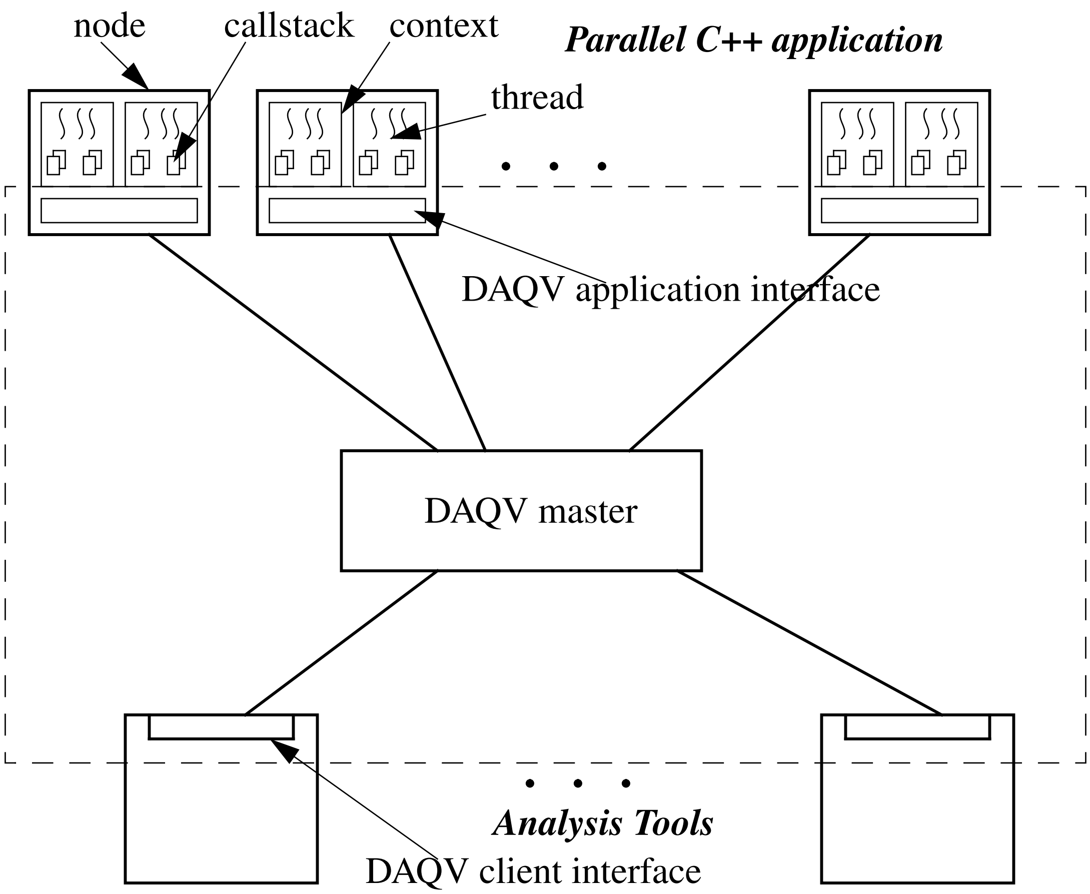
Routine Access and Data Display (RACY)



TAU Dynamic Callstack Sampling

A performance callstack is a program's callstack annotated with performance metrics. A view of a performance callstack can be generated by sampling it at runtime. The performance callstack view is defined only for those routines in the calling stack of a thread at the sample point.

Architecture

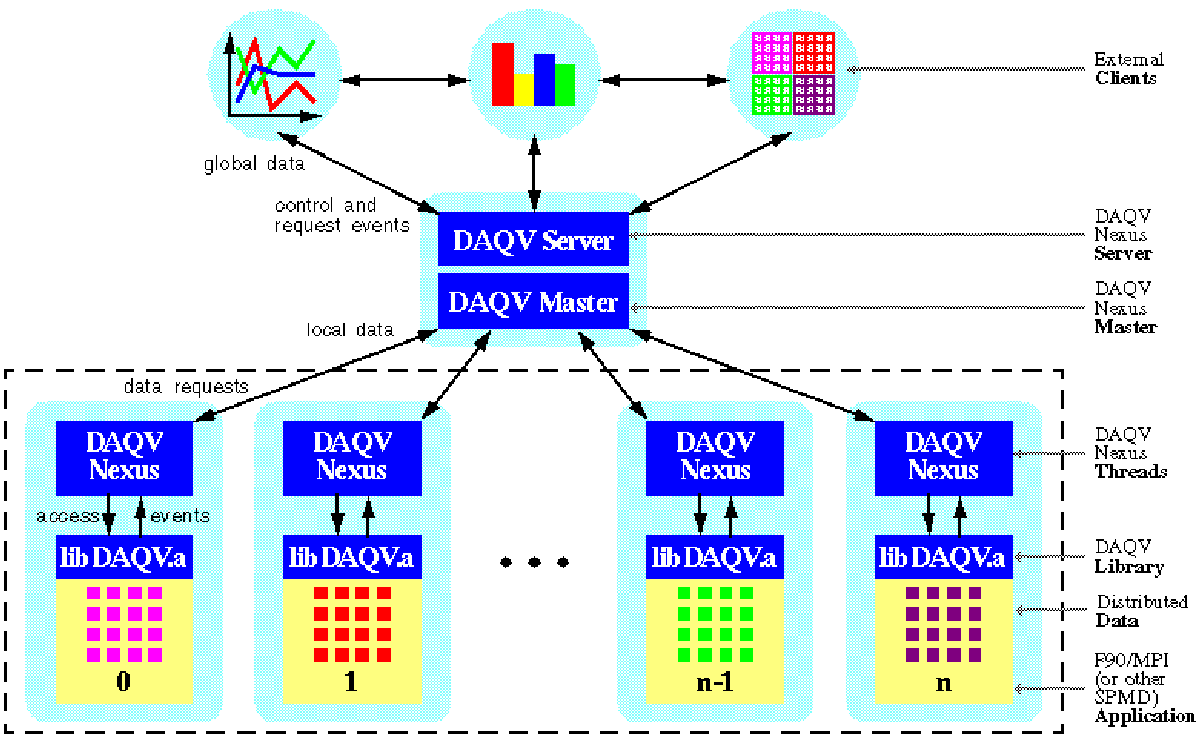


DAQV-II

(Distributed Array Query Visualization)

- Simplifies access to “global” distributed arrays in SPMD programs (e.g., Fortran 90 + MPI)
- External clients make requests for global data
- Application processes allow data access to occur at instrumentation points
- DAQV master process coordinates client requests, data access and data transfer
- Portable - uses Nexus runtime system
- DAQV-II API for building client data visualizers

DAQV-II Architecture



DAQV-II usage

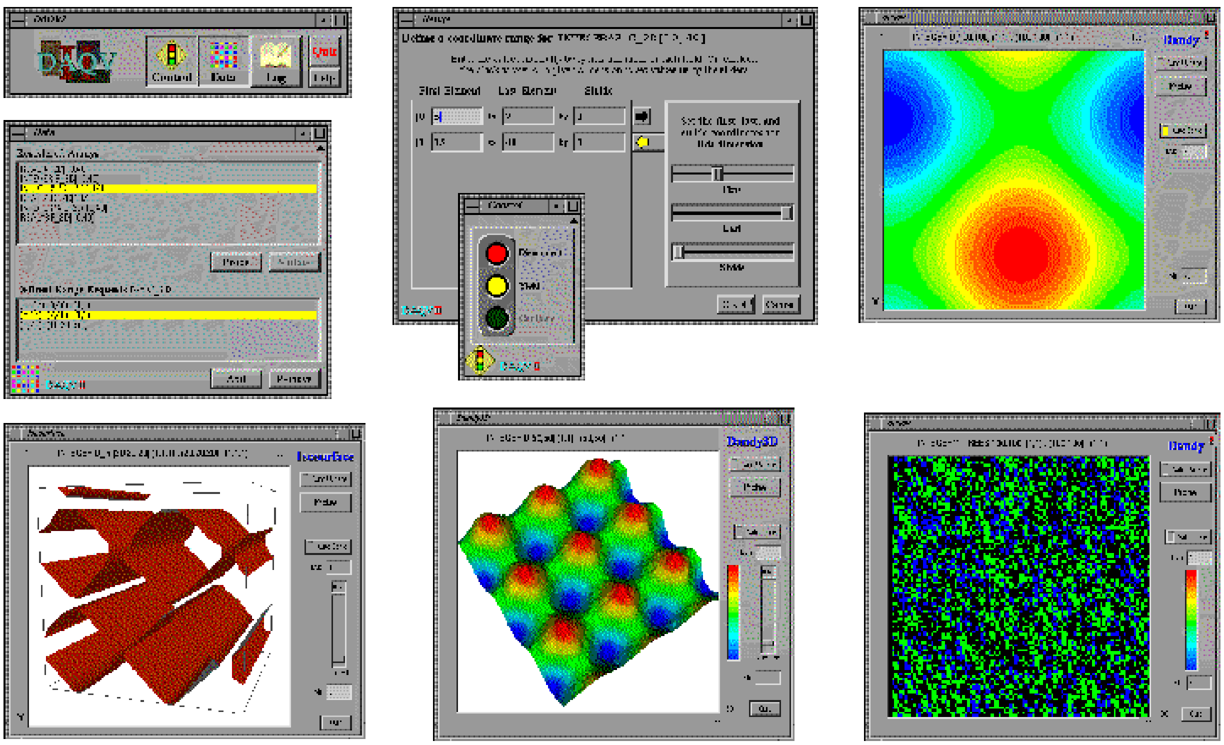
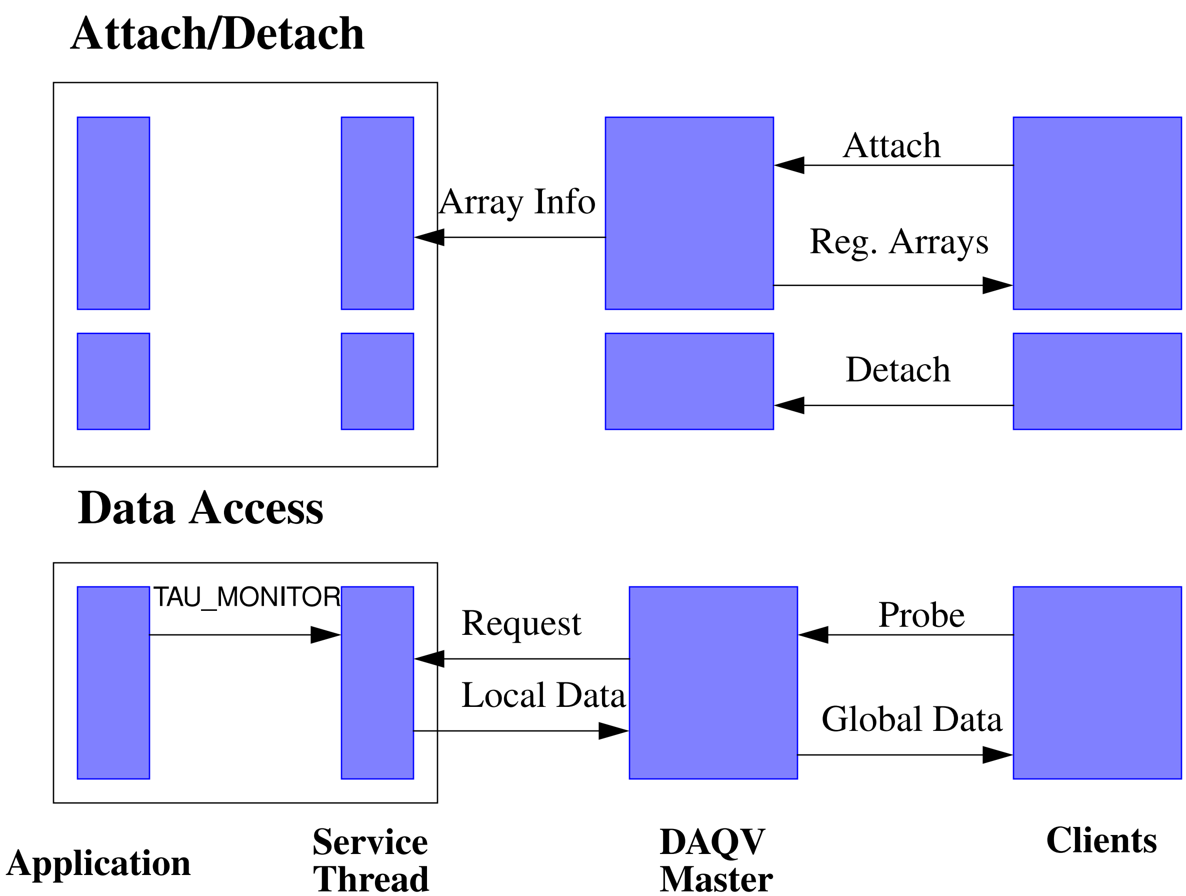


Figure 5. DAQV-II includes a general control client that can be extended to launch other clients for visualizing (or mutating) program data.

Models of Interaction

- Strict client/server model (excessive synchronization, clients difficult to implement
- Push/Pull (DAQV) (perturbation due to synchronization)
- Probe/Mutate (DAQV-II) (allows asynchronous access)
- Advantages of Probe over Client/Server synchronization
- Suitability of DAQV-II for callstack monitoring
    - allows asynchronous access
    - minimizes application synchronization overhead
    - separate thread of control for servicing data requests
    - DAQV master allows multiple client tools to access "global" performance callstack

DAQV-II Protocols



Load Monitoring with DAQV-II

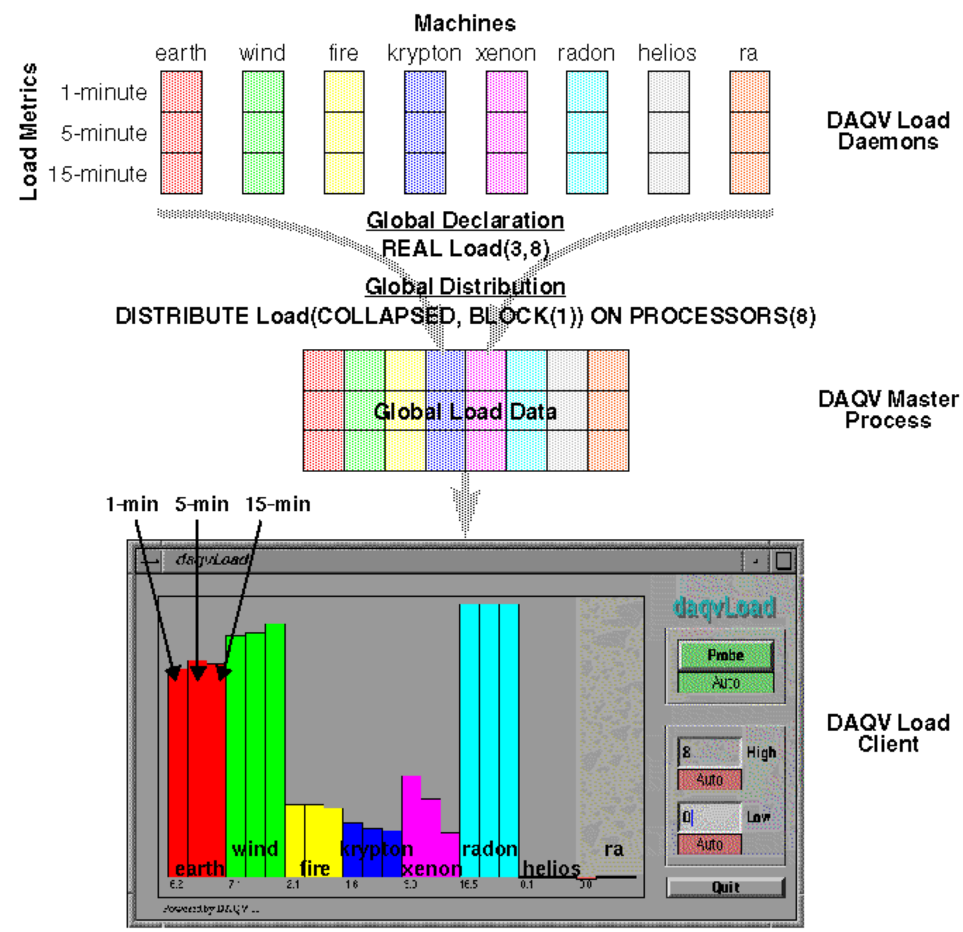
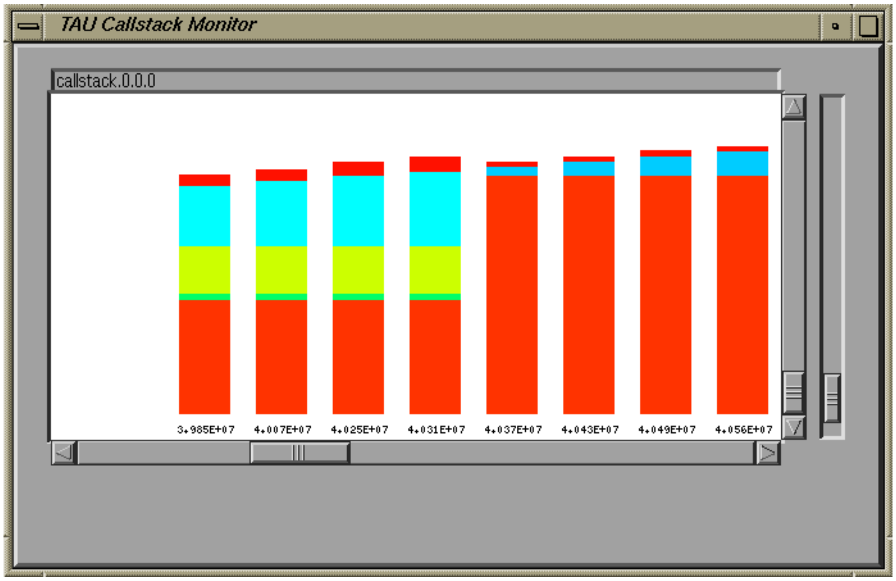
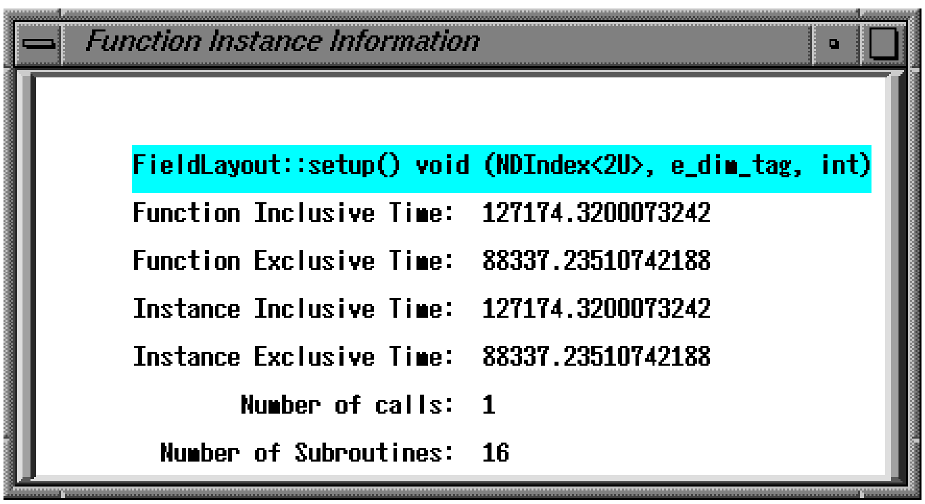
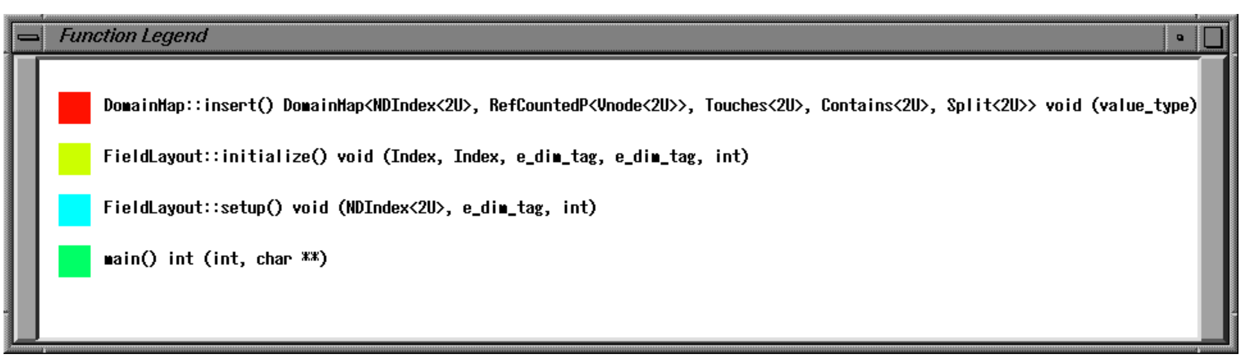


Figure 8. A distributed load monitor implemented in C and using DAQV-II.

Callstack Trace Monitor







Tuning and Analysis Utilities

Conclusions

- Callstack Sampling can be used to monitor the performance of a parallel program
- Demonstrated a portable, configurable approach to construction of a callstack monitor based on sampling
- Extended DAQV-II to support asynchronous data access

What's next...

- Build TAU clients to display performance callstack data
- Extend TAU profiling data available through DAQV to include all exclusive/inclusive function/basic block data