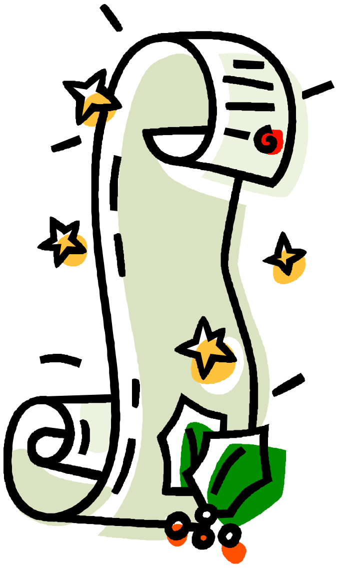
GASP: A Performance Tool Interface for Global Address Space Languages & Libraries

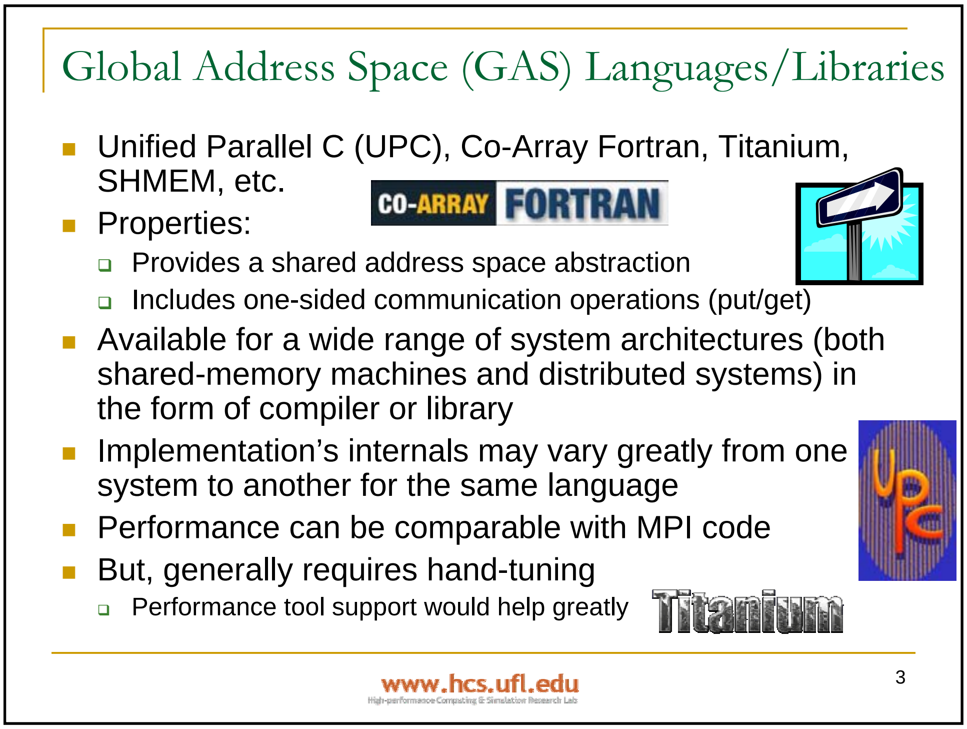
Adam Leko[1], Dan Bonachea[2], Hung-Hsun Su[1],
Bryan Golden[1], Hans Sherburne[1], Alan D. George[1]

[1] Electrical Engineering Dept., University of Florida
[2] Computer Science Dept., UC Berkeley

Outline

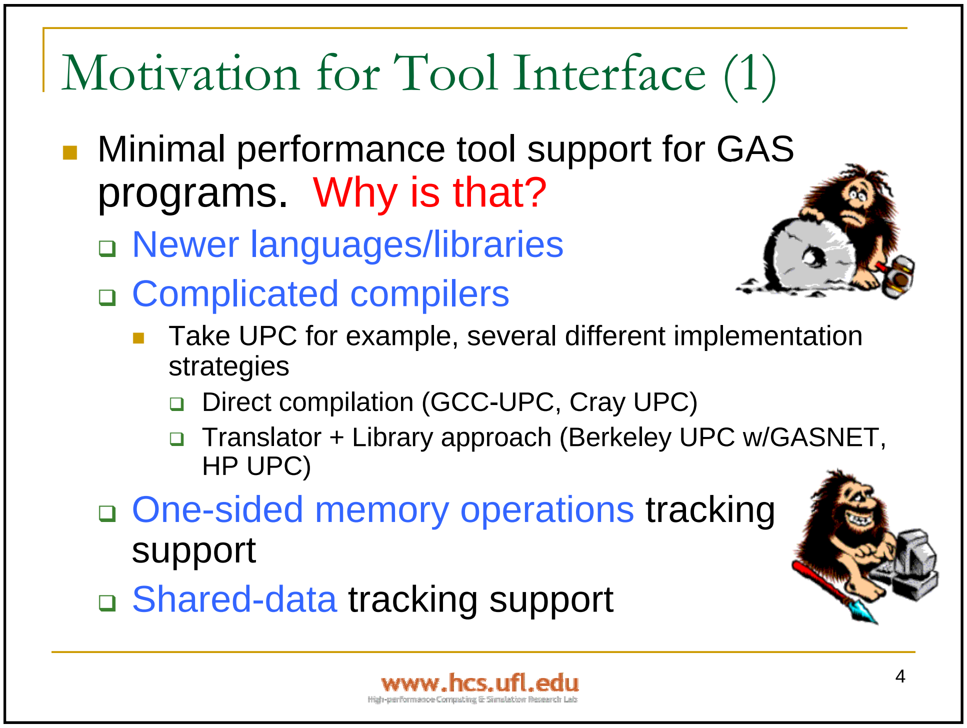
- Introduction
- GASP Overview
- GASP Interface
- GASP Overhead
- Conclusions
- Future Directions





Global Address Space (GAS) Languages/Libraries

- Unified Parallel C (UPC), Co-Array Fortran, Titanium, SHMEM, etc.
- Properties:
  - Provides a shared address space abstraction
  - Includes one-sided communication operations (put/get)
- Available for a wide range of system architectures (both shared-memory machines and distributed systems) in the form of compiler or library
- Implementation's internals may vary greatly from one system to another for the same language
- Performance can be comparable with MPI code
- But, generally requires hand-tuning
  - Performance tool support would help greatly



Motivation for Tool Interface (1)

- Minimal performance tool support for GAS programs. Why is that?
  - Newer languages/libraries
  - Complicated compilers
    - Take UPC for example, several different implementation strategies
      - Direct compilation (GCC-UPC, Cray UPC)
      - Translator + Library approach (Berkeley UPC w/GASNET, HP UPC)
  - One-sided memory operations tracking support
  - Shared-data tracking support

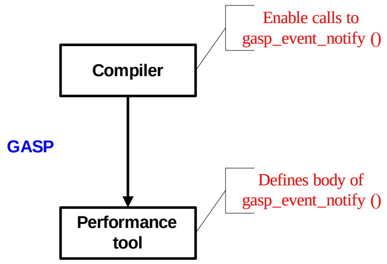
Motivation for Tool Interface (2)

- Performance tool support strategies
  - Direct source instrumentation
    - May prevent compiler optimizations/reorganization
    - How to deal with relaxed memory model?
  - Binary instrumentation
    - Not available on some architectures
    - Difficult to relate back to source code
  - Intermediate libraries
    - Wrappers for functions/procedures
    - Does not work for “pure compilers”
  - Performance interface
    - Defines basic interaction between compiler and performance tool
    - Up to compiler developers to decide how to best incorporate the interface (wrapper, translation, etc.)

Overview (1)

- **Global Address Space Performance (GASP) interface**
- *Event-based* interface
  - GAS compiler/runtime communicate with performance tools using standard interface
  - Performance tool is notified when particular actions happen at runtime
  - Implementation-agnostic
- **Notification structure**
  - Function “***callback***” to tool developer code
  - Use a ***single function name (gasp_event_notify)***
    - Pass in event ID and source code location
    - Use `varargs` for rest of arguments (like `printf`)
  - Notifications can come from compiler/runtime (system events) or from code (user events)
  - Allows calls to the source language/library to make model-specific queries

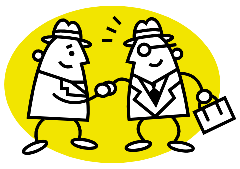
Overview (2)


Compiler
GASP
Performance tool
Enable calls to gasp_event_notify ()
Defines body of gasp_event_notify ()

```
enum gasp_event_type {gasp_event_type_start, gasp_event_type_end,
                      gasp_event_type_atomic};

void gasp_event_notify(
  unsigned int event_id,
  enum gasp_event_type event_type,
  const char* source_file,
  unsigned int source_line,
  unsigned int source_col,
  ...);
```

System Event Types (1)

- All system events have symbolic names defined in *gasp_[language].h*
- Startup and shutdown
  - Initialization called by each process after GAS runtime has been initialized
  - Exit called before all threads stop (two types of events: collective exit & non-collective exit)
- Synchronization
  - Fence, notify, wait, barrier start/end
  - Lock functions



System Event Types (2)

- Work sharing
  - Forall start/end
- Collective events
  - Broadcast, scatter, gather, etc.
- Shared variable access
  - Direct (through variable manipulations)
  - Indirect (through bulk transfer functions)
  - Non-blocking operations
- User functions
  - Beginning and end of desirable user functions

User-Defined Event Type

- Allow user to give context to performance data
- Can be used for
  - Instrumenting individual ***loops*** **or** ***regions*** in user code
  - Phase profiling
  - Hand instrumentation
- Simple ***language-independent*** API
  - gasp_create_event() creates an event with a description
  - gasp_event_start(), gasp_event_end() notify tool of region entry/exit
  - Event start/end functions also take a variable number of arguments (printf-style display) inside performance tool

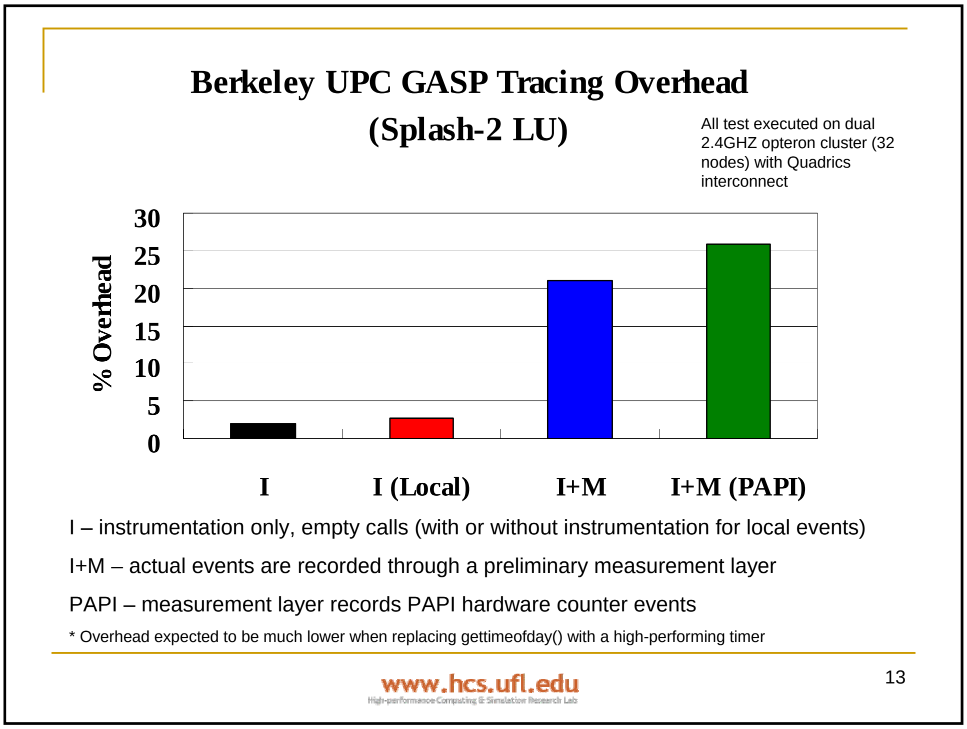
Instrumentation & Measurement Control

- Provide finer instrumentation and measurement control
- **`--profile`** flag
  - Instructs compiler to instrument all events for use with performance tool
  - Compiler should instrument all events, except
    - Shared local accesses
    - Accesses that have been privatized through optimizations

- **`--profile-local`** flag
  - Instruments everything as in --profile, but also includes shared local accesses
- **`#pragma pupc [on / off]`** directive
  - Controls instrumentation during compile time, only has effect when **`--profile`** or **`--profile-local`** have been used
  - Instructs compiler to avoid instrumentation for specific regions of code, if possible
- **`pupc_control(int on);`** function call
  - Controls measurement during runtime done by performance tool

Vendor Support

- UPC
  - Berkeley UPC
    - GASP implemented within runtime library
    - Supported with Berkeley UPC 2.4+
      - **`--enable-profile`** configure-time option
  - HP UPC
    - HP verbally agreed to support GASP at UPC '05 workshop
    - Unfortunately, GASP work has been pushed back at the moment
  - Cray UPC, MuPC, GCC-UPC
    - GASP support pending
- Others
  - Titanium GASP support is in the pipeline
  - Support for other languages/libraries pending



Berkeley UPC GASP Tracing Overhead (Splash-2 LU)

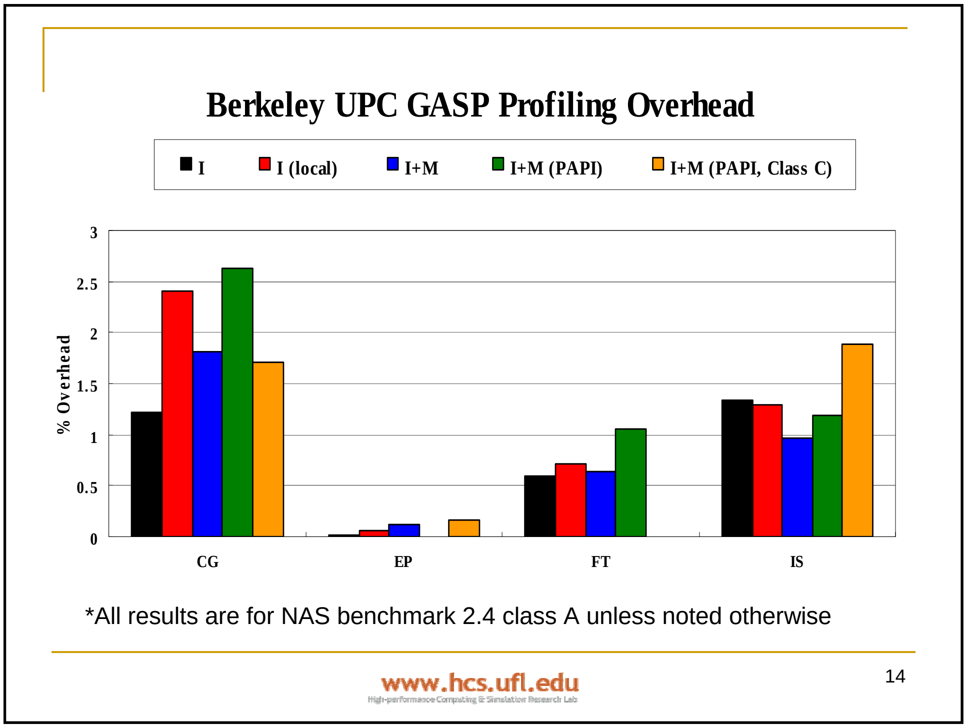
All test executed on dual 2.4GHZ opteron cluster (32 nodes) with Quadrics interconnect

I – instrumentation only, empty calls (with or without instrumentation for local events)

I+M – actual events are recorded through a preliminary measurement layer

PAPI – measurement layer records PAPI hardware counter events

* Overhead expected to be much lower when replacing gettimeofday() with a high-performing timer

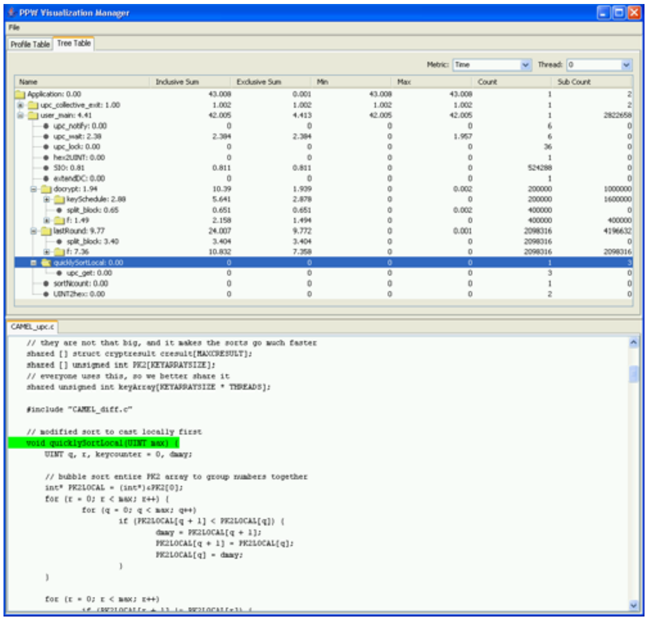


Berkeley UPC GASP Profiling Overhead

*All results are for NAS benchmark 2.4 class A unless noted otherwise

Conclusions

- GASP specifies a standard event-based performance interface for GAS languages/libraries
- Preliminary version of GASP includes UPC support (w/ low overhead, working implementation available for Berkeley UPC)
- Performance interface should be an integral part of a language/library design & implementation effort
  - Fairly straightforward for compiler developers to add support
  - Avoid interference with compiler optimization
  - But how do we get language/library implementer's support?
- GASP is integrated with a new performance analysis tool (Parallel Performance Wizard) we are currently developing for UPC and SHMEM
- Specifications for other GAS languages/libraries are forthcoming
  - May even extend beyond GAS languages/libraries to include other parallel languages/libraries such as OpenMP, MPI-2, X10, Fortress, Chapel, etc.
- For more info, see
  - http://docs.hcs.ufl.edu/upc/gasp/
  - http://docs.hcs.ufl.edu/upc/gasp/ChangeLog



Metric: Time Thread: 0

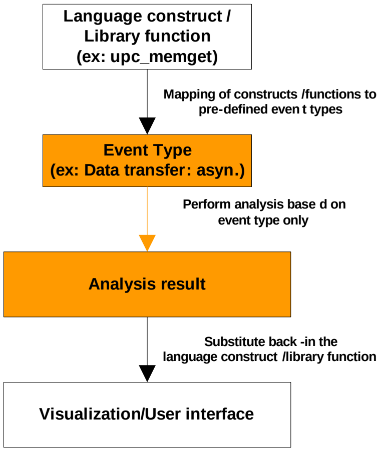
Name	Inclusive Sum	Exclusive Sum	Min	Max	Count	Sub Count
Application: 0.00	43.008	0.001	43.008	43.008	1	2
upc_collective_exit: 1.00	1.002	1.002	1.002	1.002	1	2
user_main: 4.41	42.005	4.413	42.005	42.005	1	2822658
upc_notify: 0.00	0	0	0	0	6	0
upc_wait: 2.38	2.384	2.384	0	1.957	6	0
upc_lock: 0.00	0	0	0	0	36	0
hex2UINT: 0.00	0	0	0	0	1	0
SIO: 0.81	0.811	0.811	0	0	524288	0
extendDC: 0.00	0	0	0	0	1	0
docrypt: 1.94	10.39	1.939	0	0.002	200000	1000000
keySchedule: 2.88	5.641	2.878	0	0	200000	1600000
split_block: 0.65	0.651	0.651	0	0.002	400000	0
f: 1.49	2.158	1.494	0	0	400000	400000
lastRound: 9.77	24.007	9.772	0	0.001	2098316	4196632
split_block: 3.40	3.404	3.404	0	0	2098316	0
f: 7.36	10.832	7.358	0	0	2098316	2098316
quicklySortLocal: 0.00	0	0	0	0	1	3
upc_get: 0.00	0	0	0	0	3	0
sortNcount: 0.00	0	0	0	0	1	0
UINT2hex: 0.00	0	0	0	0	2	0

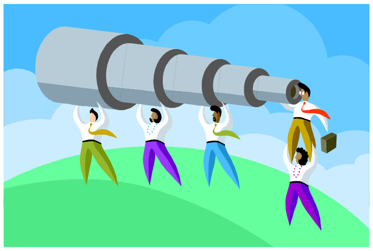
CAMEL_upc.c

```
// they are not that big, and it makes the sorts go much faster
shared [] struct cryptresult cresult[MAXRESULT];
shared [] unsigned int PK2[KEYARRAYSIZE];
// everyone uses this, so we better share it
shared unsigned int keyArray[KEYARRAYSIZE * THREADS];

#include "CAMEL_diff.c"

// modified sort to cast locally first
void quicklySortLocal(UINT max) {
    UINT q, r, keycounter = 0, dumy;

    // bubble sort entire PK2 array to group numbers together
    int* PK2LOCAL = (int*)&PK2[0];
    for (r = 0; r < max; r++) {
            for (q = 0; q < max; q++)
                    if (PK2LOCAL[q + 1] < PK2LOCAL[q]) {
                            dumy = PK2LOCAL[q + 1];
                            PK2LOCAL[q + 1] = PK2LOCAL[q];
                            PK2LOCAL[q] = dumy;
                    }
    }

    for (r = 0; r < max; r++)
            if (PK2LOCAL[r + 1] != PK2LOCAL[r]) {
```

Future Directions

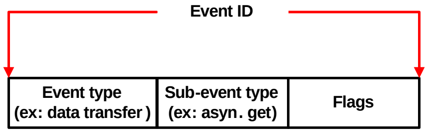
- GASP serves as a starting point for generic parallel performance analysis
- We are currently investigating the possibility of a generic parallel performance analysis approach that deals with event types rather than language constructs/library functions
  - Execution model does not differ significantly between parallel languages/libraries
  - Once a generic set of analyses is developed, it should be applicable to all languages/libraries
  - Adding performance analysis support for a new language/libraries simplifies to
    - Enabling instrumentation and measurement of events (i.e. GASP)
    - Creating a mapping of language constructs/library functions to event types
    - Small modification to visualizations to better present the result
  - Analysis of program involving multiple languages/libraries is possible


Language construct / Library function (ex: upc_memget)
Mapping of constructs /functions to pre-defined even t types
Event Type (ex: Data transfer: asyn.)
Perform analysis base d on event type only
Analysis result
Substitute back -in the language construct /library function
Visualization/User interface



🟧 Programming model independent

□ Programming model specific



Event ID

Event type (ex: data transfer)	Sub-event type (ex: asyn. get)	Flags

Q&A