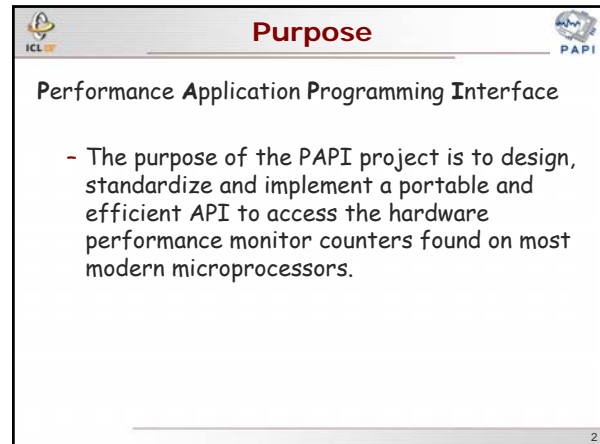


Performance Monitoring Using PAPI

CS594 Spring 2005
Dr. Dan Terpstra

INNOVATIVE COMPUTING LABORATORY
UNIVERSITY OF TENNESSEE
DEPARTMENT OF COMPUTER SCIENCE

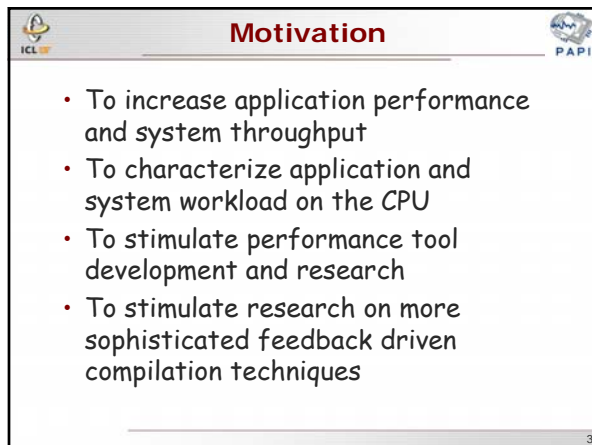
PAPI INSIDE



Purpose

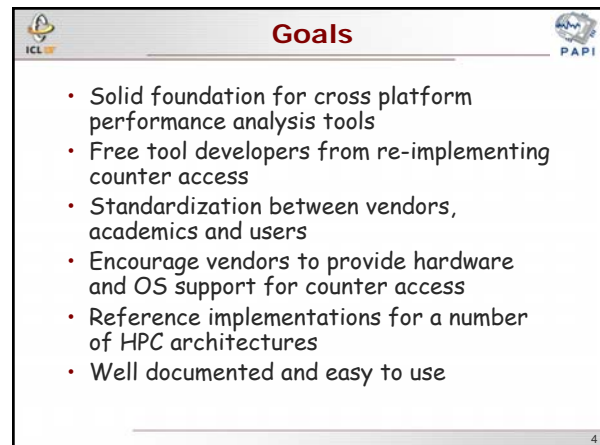
Performance Application Programming Interface

- The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.




Motivation

- To increase application performance and system throughput
- To characterize application and system workload on the CPU
- To stimulate performance tool development and research
- To stimulate research on more sophisticated feedback driven compilation techniques



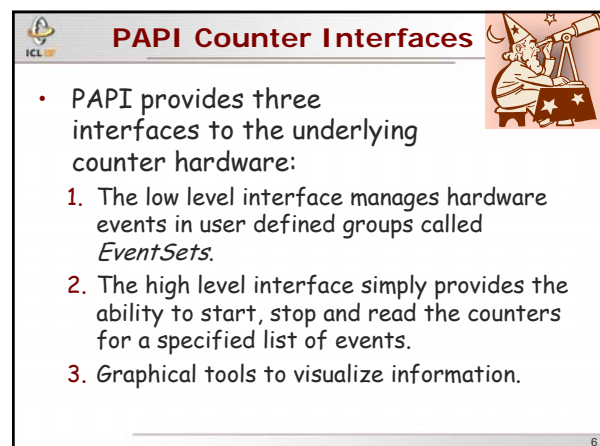
Goals

- Solid foundation for cross platform performance analysis tools
- Free tool developers from re-implementing counter access
- Standardization between vendors, academics and users
- Encourage vendors to provide hardware and OS support for counter access
- Reference implementations for a number of HPC architectures
- Well documented and easy to use



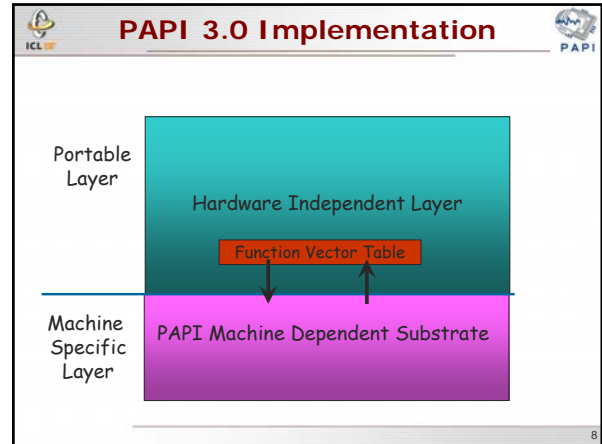
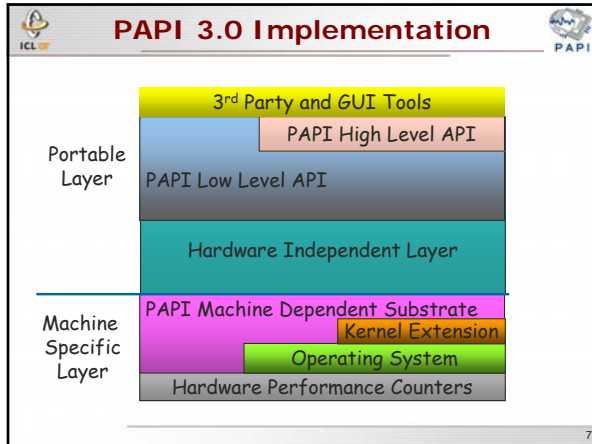
General Design of PAPI

INNOVATIVE COMPUTING LABORATORY
UNIVERSITY OF TENNESSEE
DEPARTMENT OF COMPUTER SCIENCE



PAPI Counter Interfaces

- PAPI provides three interfaces to the underlying counter hardware:
 1. The low level interface manages hardware events in user defined groups called *EventSets*.
 2. The high level interface simply provides the ability to start, stop and read the counters for a specified list of events.
 3. Graphical tools to visualize information.



- ### PAPI 3.0.8.1 Release
- Released March, 2005
 - Current Platforms
 - x86 thru Pentium 4
 - AMD Athlon, Opteron
 - Itanium Linux, Altix
 - Sun Solaris/Ultra 2.8
 - IBM AIX 4.3, 5.x / POWER3, POWER4
 - SGI IRIX/MIPS
 - Cray X1
 - Shipping on Cray XD1, XT3!
 - IBM Blue Gene / L
 - Possible Future Platforms
 - IBM PowerPC ??
 - MacOS G5 ??
 - <your cpu here>

- ### PAPI Language Interfaces
- C and Fortran bindings
 - Java
 - Lisp
 - Matlab wrappers
 - To download software:
 - <http://icl.cs.utk.edu/projects/papi/>

- ### PAPI High-level API
- Meant for application programmers wanting coarse-grained measurements
 - Calls the lower level API
 - Allows only PAPI preset events
 - Easier to use and less setup (additional code) than low-level
 - 8 API Calls:

PAPI_start_counters	PAPI_num_counters
PAPI_read_counters	PAPI_flops
PAPI_stop_counters	PAPI_flips
PAPI_accum_counters	PAPI_ipc

PAPI High-level Example

```

long long values[NUM_EVENTS];
unsigned int
Events[NUM_EVENTS]={PAPI_TOT_INS,PAPI_TOT_CYC};
/* Start the counters */
PAPI_start_counters((int*)Events,NUM_EVENTS);
/* What we are monitoring... */
do_work();
/* Stop the counters and store the results in
values */
retval = PAPI_stop_counters(values,NUM_EVENTS);
    
```

PAPI Low-level Interface

- Increased efficiency and functionality over the high level PAPI interface
- About 60 functions
(http://icl.cs.utk.edu/projects/papi/files/html_man3/papi.html-4)
- Provides information about the executable, the memory and the hardware
- Thread-safe
- Fully programmable (native events)
- Multiplexing
- Callbacks on counter overflow
- Profiling

13

Low-level Functionality

- Library initialization
PAPI_library_init, PAPI_thread_init,
PAPI_multiplex_init, PAPI_shutdown
- Timing functions
PAPI_get_real_usec, PAPI_get_virt_usec
PAPI_get_real_cyc, PAPI_get_virt_cyc
- Inquiry functions
PAPI_get_executable_info,
PAPI_get_hardware_info, PAPI_get_memory_info
- Management functions
- Simple lock
PAPI_lock/PAPI_unlock

14

Simple Low-Level Example

```
#include "papi.h"
#define NUM_EVENTS 2
int Events[NUM_EVENTS]={PAPI_FP_INS,PAPI_TOT_CYC}, EventSet;
long_long values[NUM_EVENTS];
/* Initialize the Library */
retval = PAPI_library_init(PAPI_VER_CURRENT);
/* Allocate space for the new eventset and do setup */
retval = PAPI_create_eventset(&EventSet);
/* Add flops and total cycles to the eventset */
retval = PAPI_add_events(&EventSet,Events,NUM_EVENTS);
/* Start the counters */
retval = PAPI_start(EventSet);

do_work(); /* What we want to monitor*/

/*Stop counters and store results in values */
retval = PAPI_stop(EventSet,values);
```

15

How an OS Kernel may Handle Hardware Counters

- Problems:
 - The hardware only has a small number of bits.
 - The hardware can count USER and KERNEL modes.
 - The hardware needs to be accessed by multiple users at the same time.
 - Multiple processes, threads and CPUs.
- Solution: Modify the scheduler
 - Save and accumulate the counter hardware into 64 bit virtual registers when thread/process suspends or blocks.
 - Restore the counter control register and zero the counter values when thread/process resumes.
 - Read semantics: reading the hardware counters and add them to the 64 bit quantities handled by the kernel.
- But what about single-threaded systems?...

16

PAPI Events & Event Sets



INNOVATIVE COMPUTING LABORATORY
UNIVERSITY OF TENNESSEE
DEPARTMENT OF COMPUTER SCIENCE

PAPI Preset Events

- "Standard" set of events deemed most relevant for application performance tuning
- Defined in [papiStdEventDefs.h](#)
- Mapped to native events on a given platform
 - Run `utils/avail` to see list of PAPI preset events available on a platform

18

Preset Events

- PAPI supports over 100 preset events and all platform-specific native events.
- Preset events are mappings from symbolic names to machine specific definitions for a particular hardware resource.
- Example: Total Cycles (in user mode) is **PAPI_TOT_CYC**
- PAPI also supports presets that may be derived from the underlying hardware metrics
- Example: Floating Point Instructions per Second is **PAPI_FLOPS**

Preset Listing from `utils/avail`

```

Test case avail.c: Available events and hardware information.
-----
Vendor string and code : AuthenticAMD (2)
Model string and code : AMD K8 Revision C (15)
CPU Revision          : 8.000000
CPU Megahertz         : 1992.282959
CPU's in this Node    : 2
Nodes in this System  : 1
Total CPU's           : 2
Number Hardware Counters : 4
Max Multiplex Counters : 32
-----
Name      Code      Avail  Deriv  Description (Note)
PAPI_L1_DCM 0x80000000 Yes    Yes    Level 1 data cache misses ( )
PAPI_L1_ICM 0x80000001 Yes    Yes    Level 1 instruction cache misses ( )
PAPI_L2_DCM 0x80000002 Yes    No     Level 2 data cache misses ( )
PAPI_L2_ICM 0x80000003 Yes    No     Level 2 instruction cache misses ( )
PAPI_L3_DCM 0x80000004 No     No     Level 3 data cache misses ( )
PAPI_L3_ICM 0x80000005 No     No     Level 3 instruction cache misses ( )
PAPI_L1_TCM 0x80000006 Yes    Yes    Level 1 cache misses ( )
PAPI_L2_TCM 0x80000007 Yes    Yes    Level 2 cache misses ( )
PAPI_L3_TCM 0x80000008 No     No     Level 3 cache misses ( )
.
.
.
    
```

http://icl.cs.utk.edu/projects/papi/files/html_man/papi_presets.html
<http://icl.cs.utk.edu/projects/papi/presets.html>

Native Events

- An event countable by the CPU can be counted even if there is no matching preset PAPI event
- Same interface as when setting up a preset event
- All substrates contain 'complete' native event tables
 - Preset event refer to native events
 - Native events can be accessed like presets
 - Users can enumerate both native and preset events
 - 3rd party tools can provide complete native event support
- To add a native event:


```

PAPI_event_name_to_code("PM_FPU0_FDIV", &native);
PAPI_add_event(EventSet, native);
            
```

Event sets

- The event set contains key information
 - What low-level hardware counters to use
 - Most recently read counter values
 - The state of the event set (running/not running)
 - Option settings (e.g., domain, granularity, overflow, profiling)
- Defined in [papi_internal.h](#)

Event set Operations

- Event set management


```

PAPI_create_eventset,
PAPI_add_event[s],
PAPI_rem_event[s],
PAPI_destroy_eventset
            
```
- Event set control



```

PAPI_start, PAPI_stop, PAPI_read,
PAPI_accum
            
```
- Event set inquiry


```

PAPI_query_event,
PAPI_list_events, ...
            
```

Creating an EventSet



```

integer evset, status
integer*8 values(2)
call papi_create_eventset(evset, status)
            
```

Adding events to an EventSet

evset
state: PAPI_STOPPED

1. PAPI_TOT_CYC

```
integer evset, status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
```

25

Adding events to an EventSet

evset
state: PAPI_STOPPED

1. PAPI_TOT_CYC
2. PAPI_FP_INS

```
integer evset, status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
```

26

Starting an EventSet

evset
state: PAPI_RUNNING

1. PAPI_TOT_CYC
2. PAPI_FP_INS

```
integer evset, status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
```

27

Reading an EventSet

evset
state: PAPI_RUNNING

1. PAPI_TOT_CYC
500000
2. PAPI_FP_INS
100000

```
integer evset, status
integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C do 100000 flops in 500000 cycles
call papif_read(evset, values, status)
C values contains the metrics in order of addition
C values(1) = 500000
C values(2) = 100000
```

28

Stopping an EventSet

evset
state: PAPI_STOPPED

1. PAPI_TOT_CYC
500623
2. PAPI_FP_INS
100000

```
integer evset, status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C do 100000 flops in 500000 cycles
call papif_read(evset, values, status)
C values contains the metrics in order of addition
C values(1) = 500000
C values(2) = 100000
call papif_stop(evset, values, status)
```

29

Resetting an EventSet

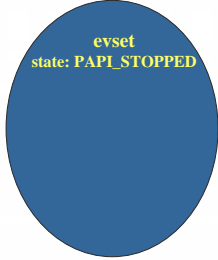
evset
state: PAPI_STOPPED

1. PAPI_TOT_CYC
0
2. PAPI_FP_INS
0

```
integer evset, status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
C do 100000 flops in 500000 cycles
call papif_read(evset, values, status)
C values contains the metrics in order of addition
C values(1) = 500000
C values(2) = 100000
call papif_stop(evset, values, status)
C state can be either RUNNING or STOPPED
C to call reset
call papif_reset(evset, status)
```

30

Emptying an EventSet



```

integer evset, status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
call papif_read(evset, values, status)
call papif_stop(evset, values, status)
call papif_reset(evset, status)
call papif_cleanup_eventset(evset, status)

```

31

Freeing an EventSet

```

integer evset, status
Integer*8 values(2)
call papif_create_eventset(evset, status)
call papif_add_event(evset, PAPI_TOT_CYC, status)
call papif_add_event(evset, PAPI_FP_INS, status)
call papif_start(evset, status)
call papif_read(evset, values, status)
call papif_stop(evset, values, status)
call papif_reset(evset, status)
call papif_cleanup_eventset(evset, status)
call papif_destroy_eventset(evset, status)

```

32

PAPI Low-Level Interface: Advanced Features



INNOVATIVE COMPUTING LABORATORY
DEPARTMENT OF COMPUTER SCIENCE

Using PAPI with Threads

- After PAPI_library_init, register a unique thread identifier function
- For Pthreads


```
retval=PAPI_thread_init(pthread_self, 0);
```
- OpenMP


```
retval=PAPI_thread_init(omp_get_thread_num, 0);
```
- Each thread is responsible for creation, start, stop and read of its own counters

34

Using PAPI with Multiplexing

- Multiplexing allows simultaneous use of more counters than are supported by the hardware.
- PAPI_multiplex_init()
 - should be called after PAPI_library_init() to initialize multiplexing
- PAPI_set_multiplex(int EventSet)
 - Used after the eventset is created to turn on multiplexing for that eventset
- Then use PAPI normally

35

Issues with Multiplexing

- Some platforms support hardware multiplexing.
- On those that don't, PAPI implements multiplexing in software.
- The more events you multiplex, the larger the sampling error in the result.

36

Callbacks on Counter Overflow

- PAPI can call user-defined handlers when an event count exceeds a specified threshold.
- Overflow can be in hardware or software.
- Hardware overflow works only on simple events; software overflow works on simple and derived events.
- For software overflow, PAPI sets up an interval timer and installs a timer interrupt handler.

37

PAPI_overflow

```
int PAPI_overflow(int EventSet, int
    EventCode, int threshold, int flags,
    PAPI_overflow_handler_t handler)
```

- Configures an EventSet to register overflows when it is PAPI_start()'d
- Multiple calls can set multiple events as overflow triggers.

38

Statistical Profiling

- PAPI supports SVR4-compatible execution profiling based on any counter event.
- PAPI_profil() creates a histogram of overflow counts for a specified region of the application code.
- Multiple events can be profiled simultaneously.
- Bin sizes can be 16-, 32-, or 64-bits.

39

PAPI_profil

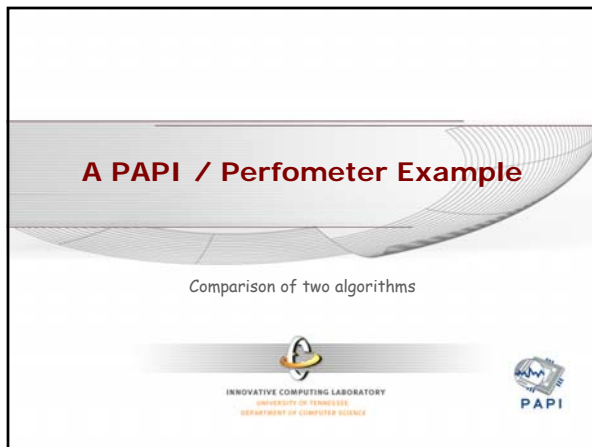
```
int PAPI_profil(unsigned short *buf, unsigned int
    bufsiz, unsigned long offset, unsigned scale,
    int EventSet, int EventCode, int threshold, int
    flags)
```

- buf - buffer of bufsiz bytes in which the histogram counts are stored
- offset - start address of the region to be profiled
- scale - contraction factor that indicates how much smaller the histogram buffer is than the region to be profiled

40

A PAPI / Perfometer Example

Comparison of two algorithms




INNOVATIVE COMPUTING LABORATORY
UNIVERSITY OF TENNESSEE
DEPARTMENT OF COMPUTER SCIENCE

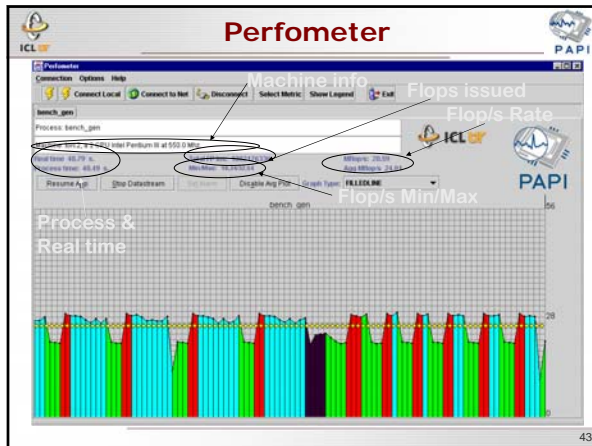
PAPI

Perfometer

- Application is instrumented with Perfometer
 - call perfometer()
 - call mark_perfometer('color')
- Application is started. At the call to **perfometer**, signal handler and timer are set to collect and send the information to a Java applet containing the graphical view.
- Sections of code that are of interest can be designated with specific colors
 - Using a call to mark_perfometer('color')
- Real-time display or trace file

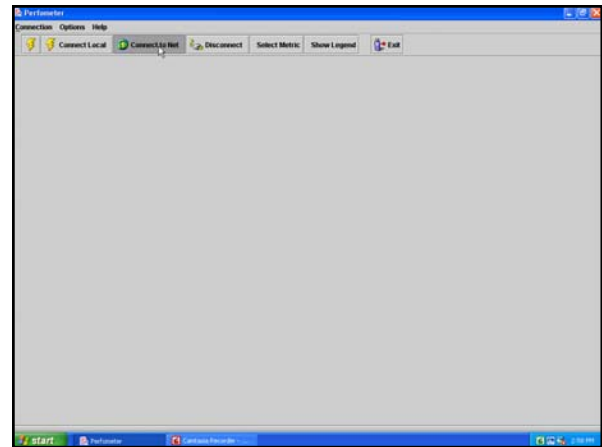


42

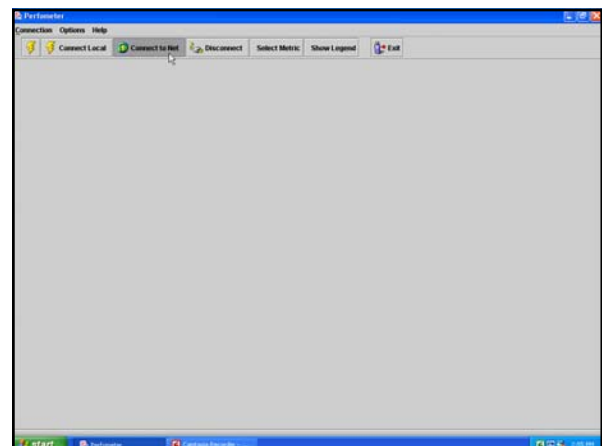


- ### A PAPI Example
- One use of PAPI might be to compare event counts in two implementations of an algorithm.
 - The following example uses PAPI to measure FLOPS for two different implementations of a matrix-matrix multiply.

- ### Reference BLAS & DGEMM
- The movie on the next slide shows Perfometer, a Java-based GUI visualizer for PAPI, monitoring the reference BLAS version of DGEMM.
 - Notice the FLOP rate for this version of DGEMM.



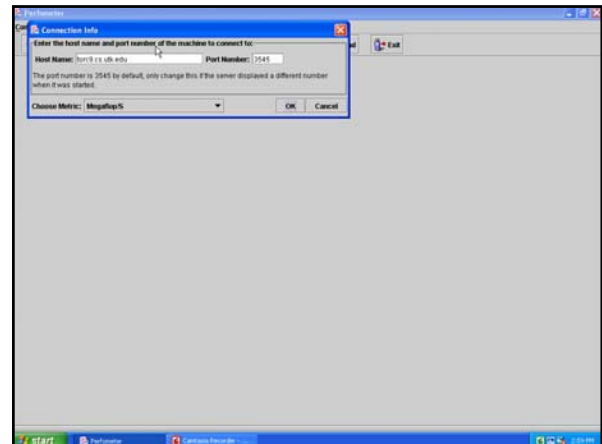
- ### ATLAS & DGEMM
- The movie on the next slide shows Perfometer monitoring the ATLAS version of DGEMM.
 - Notice how the ATLAS version of DGEMM has a much higher FLOP rate than the reference BLAS version.



DGEMM & Cache

- The ATLAS version of DGEMM showed a FLOP rate increase by greater than a factor of 3.
- Why?
- Could cache reuse be the difference?
- The next movie shows Perfometer monitoring the reference BLAS DGEMM for Level 2 cache misses.
- If there is a high rate of Level 2 cache misses then a blocking algorithm or other optimization techniques can be used to increase performance.

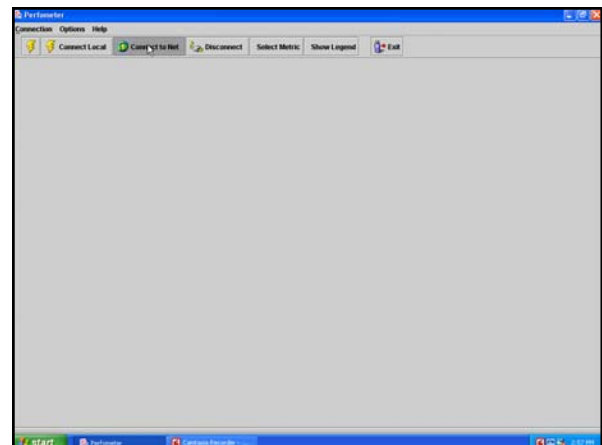
49



ATLAS & Cache

- Now compare the Level 2 cache misses of ATLAS to the previous reference BLAS version...

51



Level 2 Cache Use



- The Level 2 cache miss ratio for the reference BLAS DGEMM climbed past 50%.
- The ATLAS DGEMM had a relatively constant Level 2 cache miss ratio of around 30%.
- This shows that the ATLAS version has better memory reuse.

53

Future Work

- Function Vector Tables
 - Simpler substrate implementations
 - Vendors can implement substrates
 - Multiple simultaneous substrates
 - Substrates for other types of counters:
 - Communications; system health; special chips...
- XML Event Tables
 - Easier maintenance/modification of events
 - Better documentation capability

54

 **For More Information** 

- <http://icl.cs.utk.edu/projects/papi/>
 - Software and documentation
 - Reference materials
 - Papers and presentations
 - Third-party tools
 - Mailing lists

55