



INNOVATIVE COMPUTING  
LABORATORY

Jack Dongarra, Director



# ICL Background and History

---

- Founded in 1989
- World leader in technologies and software for scientific computing
- Has produced numerous high value tools and applications that now compose the basic fabric of high performance, scientific computing:
  - ATLAS
  - BLAS
  - FT-MPI
  - HARNESS
  - LAPACK
  - LINPACK Benchmark
  - MPI Forum
  - Netlib
  - NetSolve
  - PAPI
  - PVM
  - RIB
  - ScaLAPACK
  - Top500
- Winner of four R&D 100 awards:
  - PVM (1994)
  - NetSolve, ATLAS (1999)
  - PAPI (2001)

## Four Research Thrust Areas

---

- Numerical Linear Algebra Algorithms and Software
  - EISPACK, LINPACK, BLAS, LAPACK, ScaLAPACK, PBLAS, Templates, ATLAS, F2J
  - LAPACK/ScaLAPACK
  - LAPACK for Clusters– Matlab/Python access to Clusters
  - Exploiting single precision arithmetic
    - Cell, multicore algorithms
  - Algorithms for nano technology
  - Fault tolerant algorithms
  - Self Adapting Numerical Software; Salsa, Generic Code Optimization
- Heterogeneous Network/Grid Computing
  - PVM, MPI
  - NetSolve, FT-MPI, Open-MPI
- Software Repositories
  - NSDL, NHSE
  - RIB, ReST, Netlib, NA-DIGEST
- Performance Evaluation
  - Linpack Benchmark, Top500, PAPI, HPC Challenge, KOJAK

## PAPI, the Performance API

---

- PAPI is a middleware library that provides a consistent interface to cpu and other performance counter hardware
- Countable events are defined in two ways:
  - Platform-neutral Preset Events
  - Platform-dependent Native Events
- Events are referenced by name and collected into EventSets for measurement
- Statistical sampling is implemented by:
  - Software overflow with timer driven sampling
  - Hardware overflow if supported by the platform
- Itanium support:
  - Itanium and Itanium 2 via the perfmon/libpfm interface
  - Preliminary Montecito support available in PAPI 3.5 using Perfmon2

# PAPI Special Feature for Itanium

---

- Data/Instr Address Range Restricted Counting

- Introduced in PAPI 3.5
- Specifically supports Itanium DARR/IARR features
- Allows event monitoring on individual data structures
- Implemented via the PAPI\_set\_opt API:

```
---  
option.addr.eventset = EventSet;  
option.addr.start = (caddr_t)array;  
option.addr.end = (caddr_t)(array + size_array);  
retval = PAPI_set_opt(PAPI_DATA_ADDRESS, &option);  
actual.start = (caddr_t)array - option.addr.start_off;  
actual.end = (caddr_t)(array + size_array) + option.addr.end_off;  
---
```

- Address mapping is inexact; start\_off and end\_off provide the offsets from the requested addresses
  - Users may need to pad important data structures with dummy structures to avoid pollution of event counts
- 160 of 475 possible events can be monitored with DARR
- /utils/papi\_native\_avail identifies these events

# PAPI DARR Example

- PAPI 3.5 provides the /ctests/data\_range test case to illustrate DARR functionality
  - Measures load/store activity on 3 static and dynamic arrays, and pointers to the dynamic arrays
  - Yellow cells show deviation from theory due to overlap with adjacent dynamic arrays
  - Pink cells show overlap with other static arrays
  - Red cell shows overlap with all three pointers;
  - $115155 \approx 16384 + 3 \cdot (32768)$

|          | size (ints) | start addr | end addr | start offset | end offset | load   |          | store  |          |
|----------|-------------|------------|----------|--------------|------------|--------|----------|--------|----------|
|          |             |            |          |              |            | theory | measured | theory | measured |
| pointer1 | 1           | 11640      | 11648    | 0            | 0          | 32768  | 32768    | 0      | 0        |
| pointer2 |             | 11628      | 11630    | 0            | 0          |        | 32768    |        | 0        |
| pointer3 |             | 11638      | 11640    | 0            | 0          |        | 32768    |        | 0        |
| dynamic1 | 16384       | 4044010    | 4054010  | 10           | 0          | 16384  | 16384    | 16384  | 16384    |
| dynamic2 |             | 4054020    | 4064020  | 20           | 0          |        | 16388    |        | 16388    |
| dynamic3 |             | 4064030    | 4074030  | 30           | 0          |        | 16392    |        | 16392    |
| static1  | 16384       | 218cc      | 318cc    | 18cc         | 734        |        | 18432    | 16384  | 18432    |
| static2  |             | 118cc      | 218cc    | 18cc         | 734        |        | 115155   |        | 16845    |
| static3  |             | 318cc      | 418cc    | 18cc         | 734        |        | 17971    |        | 17971    |

# OpenMP Scalability Analysis

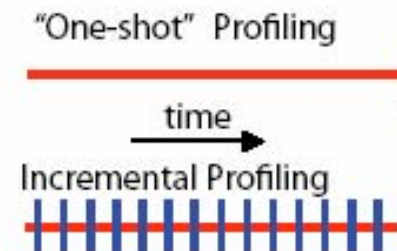
- Tool used: OpenMP profiler ompP
- Scalability analysis
  - Classify execution time into "Work" and four overhead categories: "Thread Management", "Limited Parallelism", "Imbalance", "Synchronization"
  - Analyze how overheads behave for increasing thread counts
  - Graphs show accumulated runtime over all threads for fixed workload (strong scaling)
  - Application example: 314.mgrid\_m from the SPEC OpenMP benchmark suite



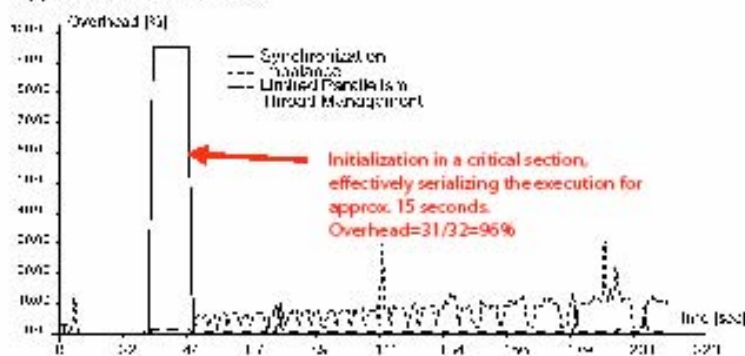
- Scaling from 2 to 32 processors on an SGI Altix machine
- Markedly smaller load imbalance for thread counts of 32 and 16. Only three parallel loops show this behavior
- In all three cases, the iteration count is always a power of two (2 to 256), hence thread counts which are not a power of two exhibit larger load imbalance

# OpenMP Incremental Profiling (1)

- Idea: combine advantages of profiling and tracing
  - Add a temporal dimension to profiling-type performance data
  - See what happens during the execution without capturing full traces
  - See when OpenMP constructs are executed
  - See when overheads arise
  - Analyze hardware counter heatmaps



Application: 328.fma3d\_m



- Application 328.fma3d\_m
- Executed on a 32-processor SGI Altix machine (Itanium 2, 1.6 GHz, 6MB L3 Cache)
- X-axis is time in seconds
- Y-axis is percent of accumulated execution time spent in one of the overhead classes

## OpenMP Incremental Profiling (2)

- Performance counter “heatmaps”
  - X-axis: time in seconds, y-axis: thread-ID
  - Color: number of hardware counter events observed for this sample
  - Example application “310.wupwise”, medium-sized variant, counter: DATA\_EAR\_CACHE\_LAT1024, events correspond roughly to remote memory accesses
  - Notice **difference** of threads {0,16} to {8,15,24,31} to the rest

