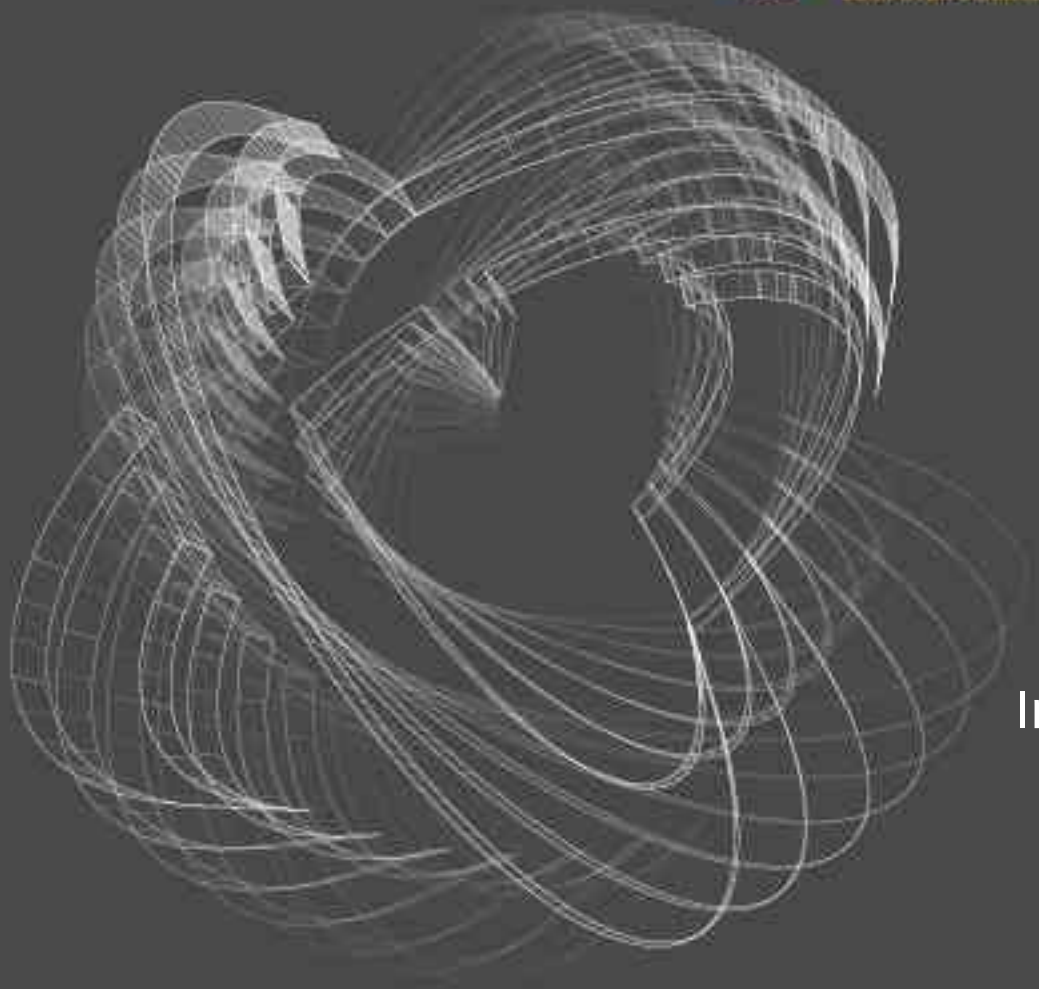


Innovative Computing Laboratory
UNIVERSITY OF TENNESSEE
COMPUTER SCIENCE DEPARTMENT

Performance Analysis of HPC Architectures

Philip J. Mucci
Shirley Moore
Innovative Computing Laboratory
University of Tennessee
mucci@cs.utk.edu

HPC User Forum
Princeton, NJ
September 16, 2003

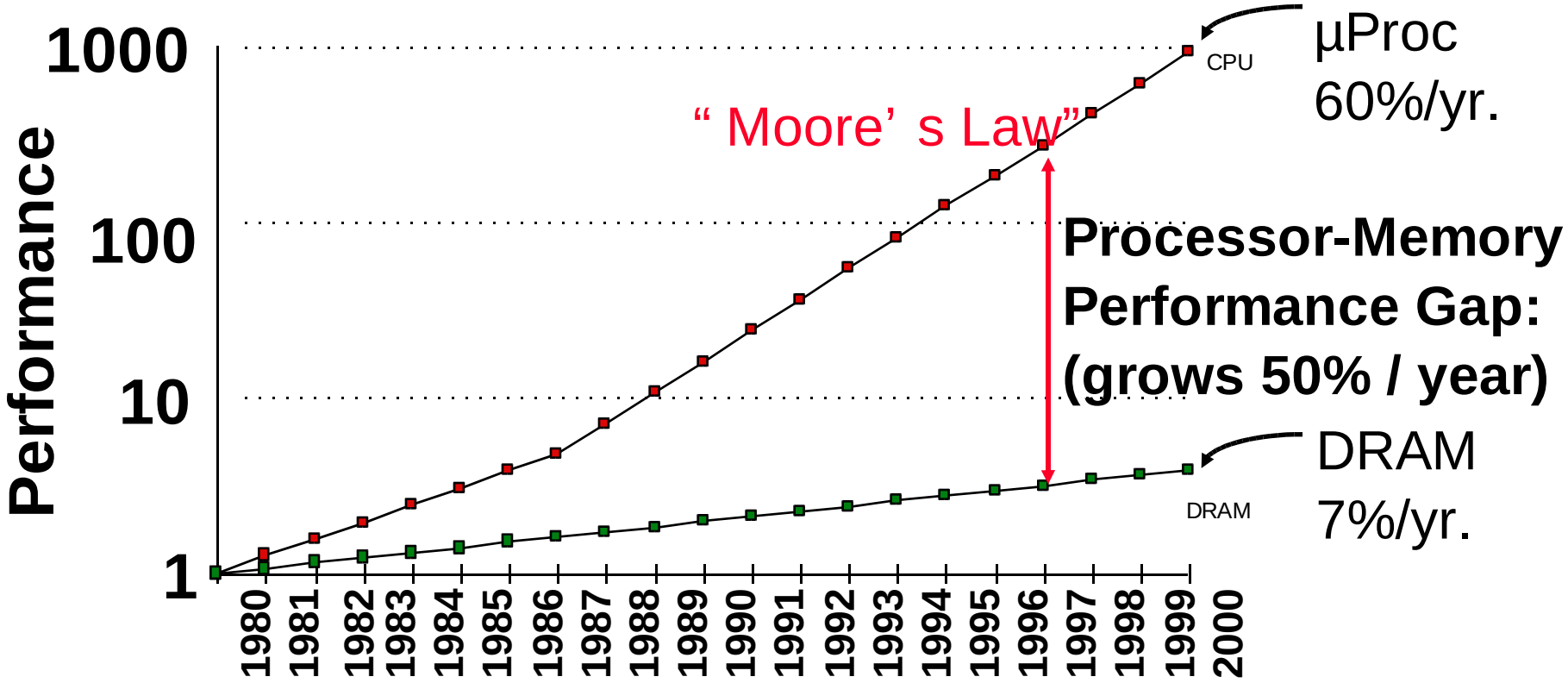
Innovative Computing Laboratory

- Relevant Research Efforts
  - Benchmarking and Evaluation
  - Performance Analysis
  - Numerical Libraries
  - Automated Optimization

Architecture Evolution

- Moore’ s Law: Microprocessor CPU performance doubles every 18 months.
- Cost and size of storage have fallen along a similar exponential curve.
- But decrease in time to access storage has not kept up leading to:
  - deeper and more complex memory hierarchies
  - “ load-store” architecture

Processor-DRAM Gap (latency)



ICL Benchmarking

- ParkBench
  - Microbenchmarks
  - Kernels
  - Applications
- LLCBench
  - Blasbench
  - Cachebench
  - Mpbench

LLCBench: Low Level Characterization Benchmarks

- Philosophy: Measure low level characteristics of the architecture and run-time system to aid in the parameterization of performance of large codes
  - Easy to run
  - Somewhat easy to understand

Cachebench: HP-ZX1 Itanium 2

Mpbench: HP-ZX1 Itanium 2

Blasbench: HP-ZX1 Itanium 2

References

http://icl.cs.utk.edu/projects/llcbench

http://www.parkbench.org

http://www.cs.uoregon.edu/research/paracomp/tau

Processor Families

- Have high-level design features in common
- Four broad families over the past 30 years
  - CISC
  - Vector
  - RISC
  - VLIW

CISC

- Complex Instruction Set Computer
- Designed in the 1970s
- Goal: define a set of assembly instructions so that high-level language constructs could be translated into as few assembly language instructions as possible => many instructions access memory, many instruction types
- CISC instructions are typically broken down into lower level instructions called *microcode*.
- Difficult to pipeline instructions on CISC processors
- Examples: VAX 11/780, Intel Pentium Pro

Vector Processors

- Seymour Cray introduced the Cray 1 in 1976.
- Dominated HPC in the 1980s
- Perform operations on vectors of data
- Vector pipelining (called *chaining*)
- Examples: Cray T90, Convex C-4, Cray SV1, Cray SX-6, Cray X1, POWER5?

RISC

- Reduced Instruction Set Computer
- Designed in the 1980s
- Goals
  - Decrease the number of clocks per instruction (CPI)
  - Pipeline instructions as much as possible
- Features
  - No microcode
  - Relatively few instructions all the same length
  - Only load and store instructions access memory
  - Execution of branch delay slots
  - More registers than CISC processors

RISC (cont.)

- Additional features
  - Branch predicition
  - Superscalar processors
    - Static scheduling
    - Dynamic scheduling
  - Out-of-order execution
  - Speculative execution
- Examples: MIPS R10K/12K/14K, Alpha21264, Sun UltraSparc-3, IBM Power3/Power4

VLIW

- Very Long Instruction Word
- Explicitly designed for instruction level parallelism (ILP)
- Software determines which instructions can be performed in parallel, bundles this information and the instructions, and passes the bundle to the hardware.
- Example: Intel-HP Itanium

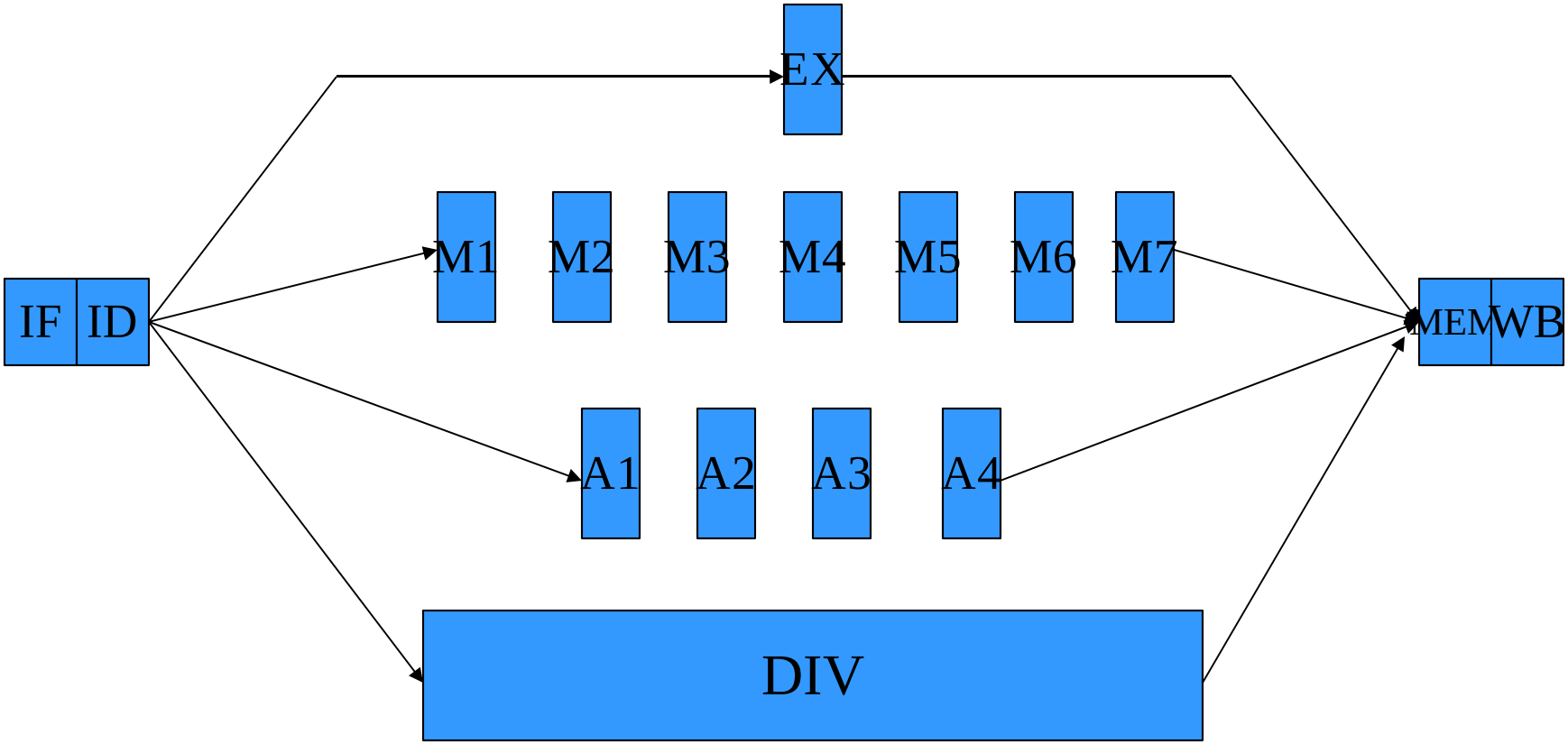
Architecture Changes in the 1990s

- 64-bit addresses
- Optimization of conditional branches via conditional execution (e.g., conditional move)
- Optimization of cache performance via prefetch
- Support for multimedia and DSP instructions
- Faster integer and floating-point operations
- Reducing branch costs with dynamic hardware prediction

Pipelining

- Overlapping the execution of multiple instructions
- Assembly line metaphor
- Simple pipeline stages
  - Instruction fetch cycle (IF)
  - Instruction decode/register fetch cycle (ID)
  - Execution/effective address cycle (EX)
  - Memory access/branch completion cycle (MEM)
  - Write-back cycle (WB)

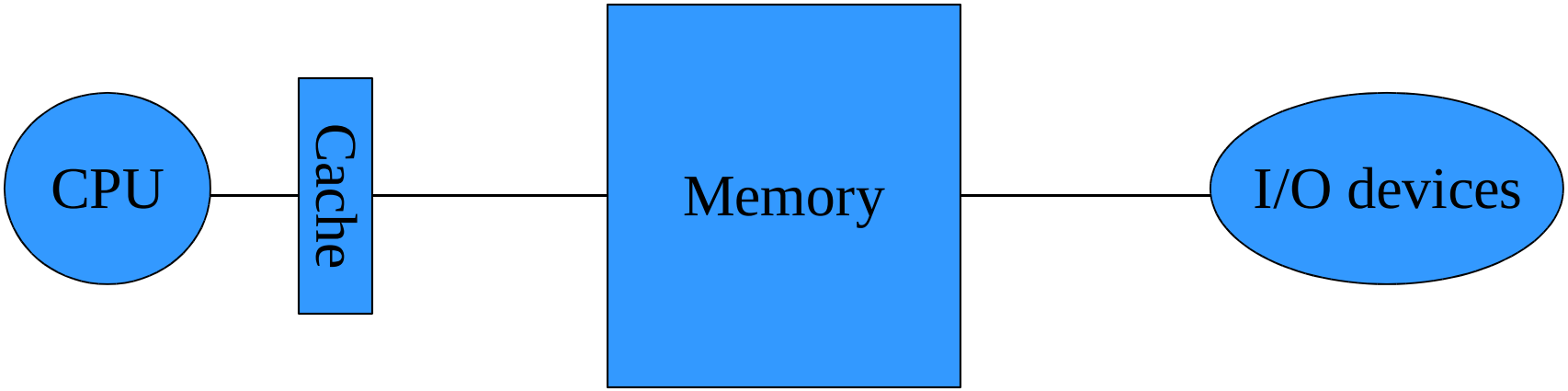
Pipeline with Multicycle Operations



Memory Hierarchy Design

- Exploits principle of *locality* – programs tend to reuse data and instructions they have used recently
  - *Temporal locality* – recently accessed items like to be accessed in the near future
  - *Spatial locality* – items whose addresses are near each other likely to accessed close together in time
- Take advantage of cost-performance of memory technologies
- Fast memory is more expensive.
- Goal: Provide a memory system with cost almost as low as the cheapest level of memory and speed almost as fast as the fastest level.

Typical Memory Hierarchy

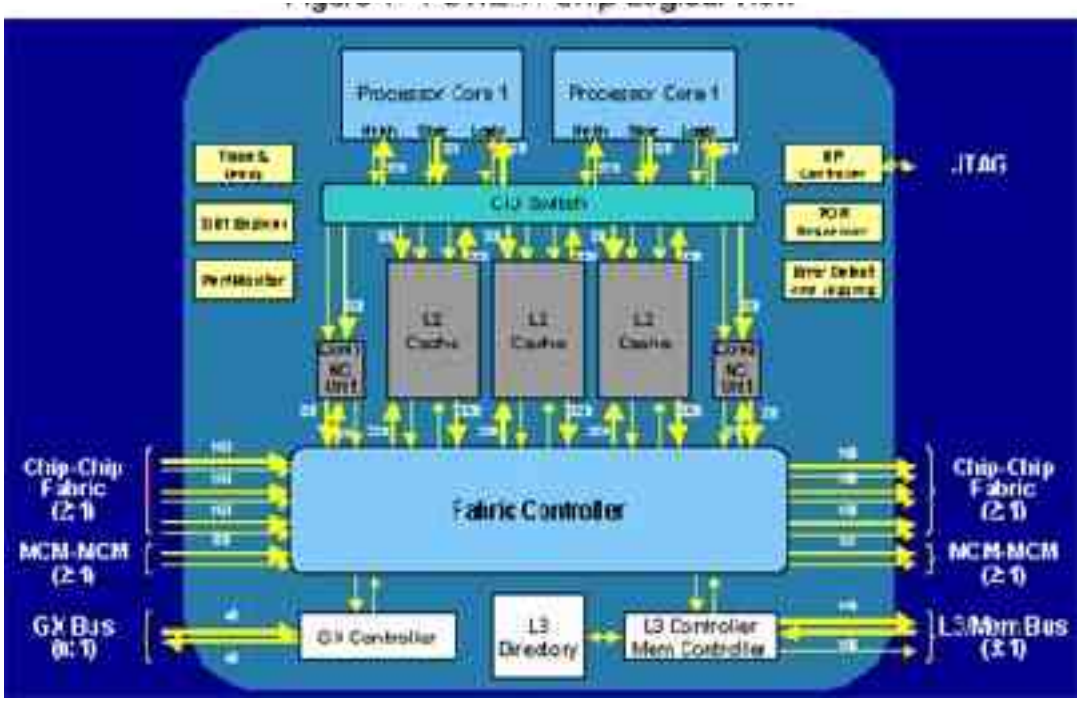


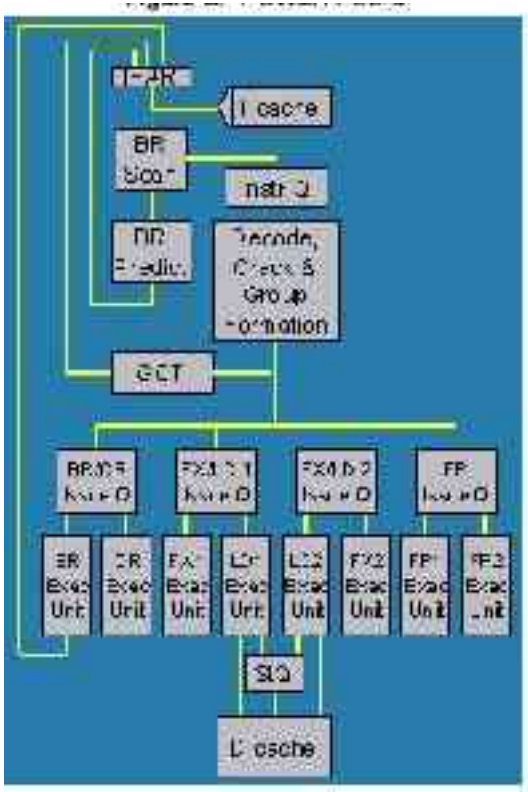
	Register reference	Cache reference	Memory reference	Disk reference
Size:	500 bytes	64 KB	512 MB	100 GB
Speed:	0.25ns	1 ns	100 ns	5 ms

Cache Characteristics

- Number of caches
- Cache sizes
- Cache line size
- Associativity
- Replacement policy
- Write strategy

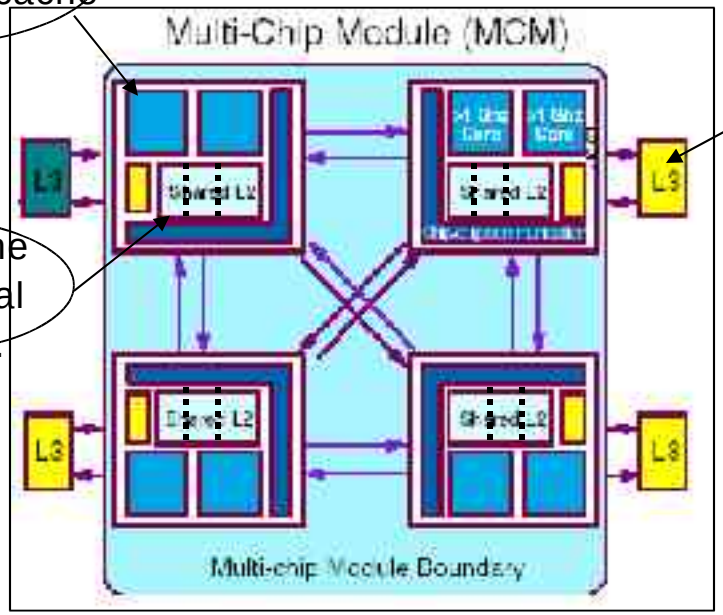
POWER4 Diagram





Node Design

1.3 GHz processor
32 KB Level 1 cache



Level 3 cache
512 MB total
128 MB ea.

Level 2 cache
1440 KB total
480 KB ea.

Four MCMs comprise one node of a regatta system.

POWER4 vs. POWER3

- 1.3 GHz POWER 4 clock rate compared to 375 MHz for POWER3
- But performance of floating-point intensive applications is typically only two to three times faster.
- More difficult to approach peak performance on POWER4 than on POWER3 because of
  - Increased FPU pipeline depth
  - Reduced L1 cache size
  - Higher latency (in terms of cycles) of the higher level caches

Application Performance

- How well does the application map onto the underlying architecture?
  - Effectiveness of compiler optimization
  - Use of the memory hierarchy
  - Keeping the pipeline full
- Application characteristics
  - Instruction mix
  - Type of memory access
  - Communication overhead

What are good Performance Measures?

- Time
- FLOPS
- IPC/CPICycles per instruction
  - Overall
  - By instruction type
  - By routine and loop

Parallel Performance - Speedup and Scalability

- *Speedup* is the ratio of the running time on a single processor to the parallel running time on N processors.

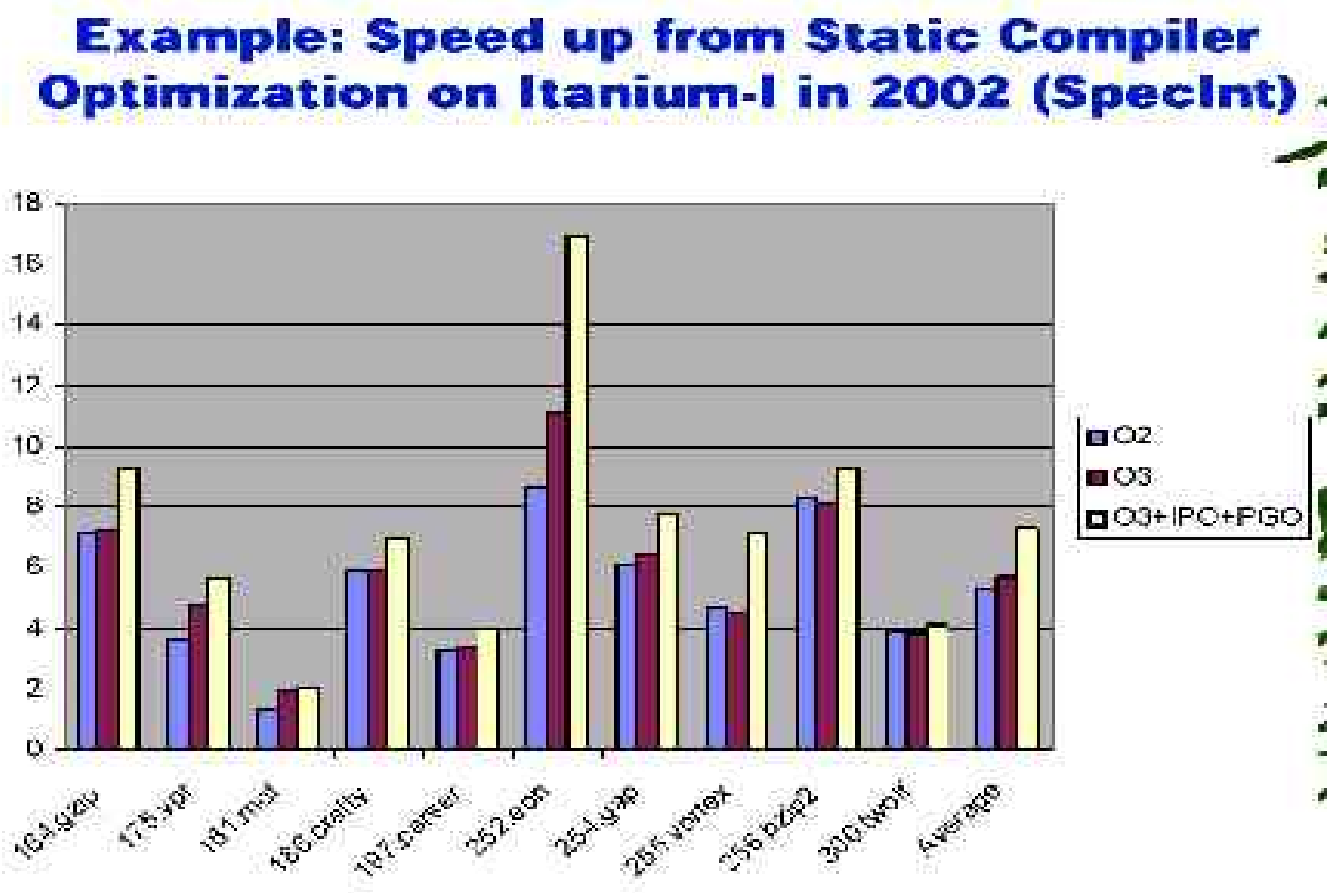
  Speedup = T(1)/T(N)

- An application is *scalable* if the speedup on N processors is close to N.
- With *scaled speedup*, an application is said to be scalable if, when the number of processors and the problem size are increased by a factor of N, the running time remains the same.

Factors that Limit Scalability

- Amdahl’ s Law
- Communication Overhead
- Load Imbalance

Importance of Optimization



Steps in Optimizing Code

- Optimize compiler switches
- Integrate high-performance libraries
- Profile code to determine where most time is being spent
- Optimize blocks of code that dominate execution time by using performance data to determine why the bottlenecks exist
- Always examine correctness and performance at every stage!

Memory Performance Measurement

- Costs of cache misses
- Average memory access time
- Average memory stalls per instruction
- Effectiveness of prefetching

Measure above using hardware counters and use results for performance modeling and analysis

Performance of Loops

- Unroll inner loops to increase the number of independent computations in each iteration to keep the pipelines full
- Unroll outer loops to increase the ratio of computation to load and store instructions so that loop performance is limited by computation rather than data movement
- Measure effectiveness of loop transformations using hardware counter data – use for hand tuning or compiler feedback

Profiling

Recording of summary information during execution

- inclusive, exclusive time, # calls, hardware statistics, …

Reflects performance behavior of program entities

- functions, loops, basic blocks
- user-defined “ semantic” entities

Very good for low-cost performance assessment

Helps to expose performance bottlenecks and hotspots

Implemented through

- sampling: periodic OS interrupts or hardware counter traps
- instrumentation: direct insertion of measurement code

Tracing

Recording of information about significant points (events) during program execution

- entering/exiting code region (function, loop, block, ..)
- thread/process interactions (e.g., send/receive message)

Save information in event record

- timestamp
- CPU identifier, thread identifier
- Event type and event-specific information

Event trace is a time-sequenced stream of event records

Can be used to reconstruct dynamic program behavior

Typically requires code instrumentation

PAPI

Performance **A**pplication **P**rogramming **I**nterface

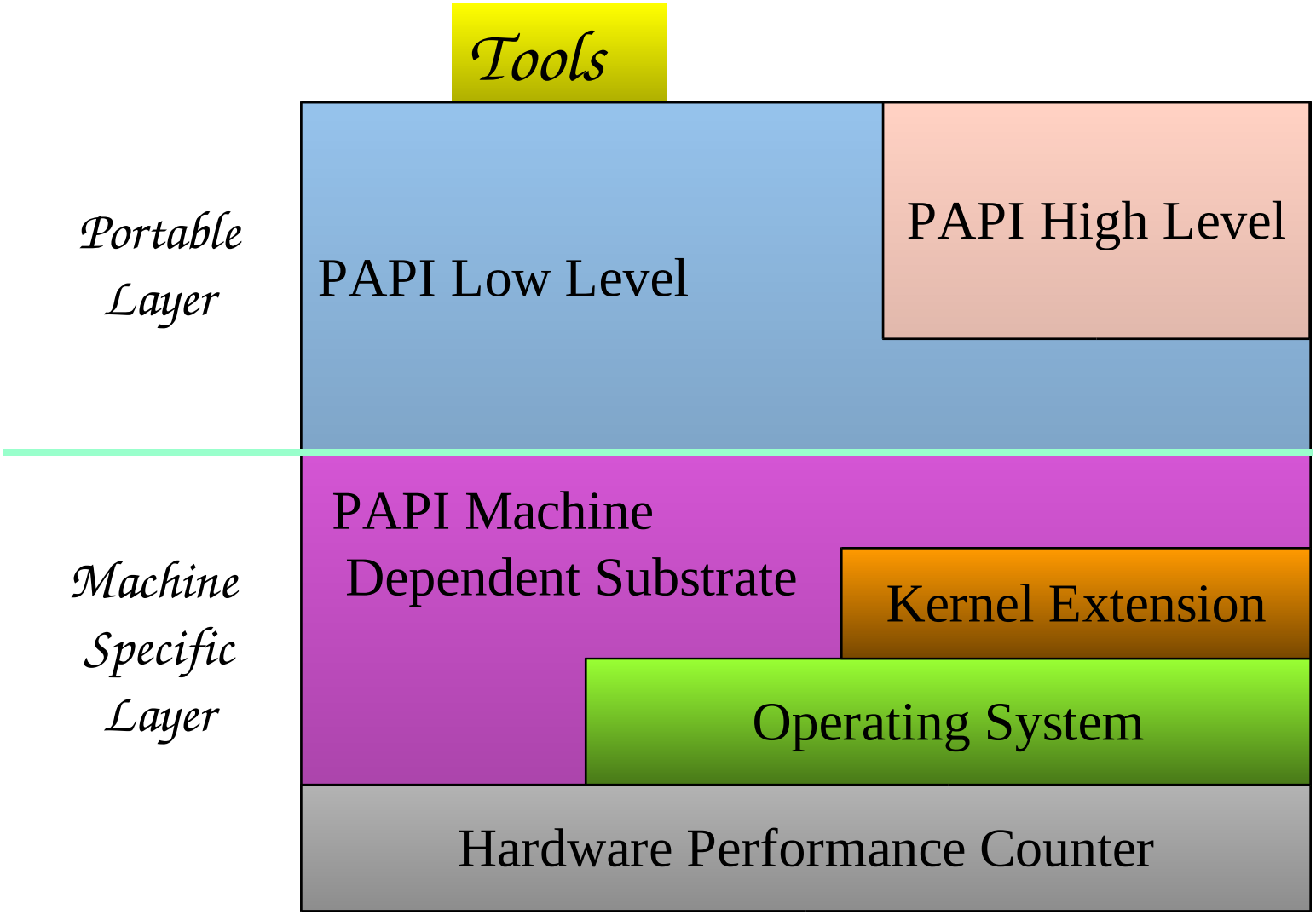
The purpose of the PAPI project is to design, standardize and implement a portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.

Parallel Tools Consortium project

In use at DOE ASCI Labs, NSF PACI sites, DoD HPC Centers

PAPI 3.0 release planned for SC' 03

PAPI Implementation



TAU: Tuning and Analysis Utilities

- Portable profiling and tracing toolkit for performance analysis of parallel programs
  - Fortran 77/90, C, C++, Java
  - OpenMP, Pthreads, MPI, mixed mode
- In use at DOE ASCI Labs and DoD HPC Centers

ATLAS

- Automatically Tuned Linear Algebra Software – AEOS techniques applied to linear algebra software, particularly the BLAS
- ATLAS-2 – generalizing ATLAS – given arbitrary code
  - Identify region for optimzation
  - Generation different version based on machine parameters
  - Empirical search for the best optimized code

AEOS

Automated Empirical Optimization of Software – generates optimized libraries quickly on new platforms

- Code generation
- Multiple implementations
- Parameterization
- Sophisticated timers
- Robust search heuristics