Programming Models for High-Performance Computing

Mary Hall
USC/ISI
June 24, 2002
PIMs: A Future Architectural Component

Class 2: (e.g., DIVA)  
Class 4: Multicomputer
Applications Well-Suited to PIMs

- **Memory bound**
  - High bandwidth requirements (e.g., data reduction)
  - Inability to make effective use of cache

- **Inherent parallelism**
  - Fine-grain parallelism exploited on chip
  - Coarse-grain parallelism across chips

- **Case-study applications**
  - Image Correlation (streaming)
  - Conjugate Gradient (sparse-matrix)
  - OO7 Object-Oriented Database Benchmark (pointer-based)
Speedups over Host Execution

![Graph showing speedups over host execution. The x-axis represents the number of PIM nodes, ranging from 1 to 32. The y-axis represents speedups, ranging from 0 to 50. The graph includes three different curves: NAS CG (yellow triangles), OO7-poll (purple stars), and TM (teal plus signs). The curves show an increasing trend as the number of PIM nodes increases.]
In the Last Decade, What is Different?

- Latencies are increasingly non-uniform
  - Gap between best- and worst-case access times growing
  - Processor-memory gap, wire latency, …

- Convergence towards clusters and MPI
  - Great for some applications/developers, not for everyone

- Programming Models and Compilers
  - Simple and evolutionary approaches most successful
  - Increasingly rely on run-time and feedback-directed optimization
Essentials of Future Systems

✲ Architecture Provides
  – Globally addressable, distributed address space
  – Tightly integrated processors, memory, communication
  – Independent threads of control

✲ Programming Model Supports
  – Specification of coarse-grain parallelism and coarse-grain spatial locality
  – Adaptation to compile-time/run-time events

✲ Compiler/Run-Time System Derives
  – Exploits fine-grain parallelism and fine-grain locality
  – Performance tuning parameters
Consider Cache & Register Locality

◆ Most scientific applications achieve less than 10% processor efficiency!
◆ Why?
  – Ineffective use of caches and registers
  – Memory accesses severely limit instruction-level parallelism
◆ Why don’t compiler optimizations solve these problems?
  – Hardware complexity growing – static prediction hard
  – Complex interactions between optimizations
  – Some decisions are input-data dependent
An Empirical Compiler

Source Code

Experiments Engine

Program Analysis & Transformations

Binary

Run-Time-System

Environment

Experience Base

Optimized Executable

Source Code Variants

Architecture Description
Summary

- Tighter integration of processor, memory, communication
- Locality plays a bigger role
  - Not just local/global, but also proximity
- Programmer expresses coarse-grain locality and coarse-grain parallelism
- Empirically-Guided Optimization
  - Compiler- and Programmer-Directed
  - Compiler derives fine-grain parallelism and locality