Keeping Programming in MIND for the MTV Petaflops Generation

Thomas Sterling

California Institute of Technology
and
NASA Jet Propulsion Laboratory
June 24, 2002
Future HPC Systems?

- Commodity Clusters
  - Beowulf, NOW, Chiba City, HP SC
- MPP
  - T3E, ASCI White
- PVP
  - SX-6, SV-2, Earth Simulator
- Hybrid Technology
  - HTMT
Petaflops in 2010 to 2012

- Flops performance gain 25X at system level
  - Clock rate > 10 GHz (chip wide)
  - The rest is ILP and SOC
  - Petaflops in < 10K chips

- Memory capacity 64X
  - Petabyte in 128K chips
  - for ~$4M
  - 7%/yr speedup in access time
  - Will take ~30 times longer to read contents of memory chip
  - Will take > 100X longer in clock cycles (not including latency)

- I/O interface will grow slowly toward 1 Tbps (maybe optics?)

- 1 Petaflops System in 2010 cost between $50M & $200M

- Power is unclear
  - Between 3 Mwatts and 25 Mwatts
Problems with Conventional Approach

- Processor bound
  - ALU and IP constrained – ALU utilization wrong metric
  - Latency intolerant
  - Parallelism exposure inadequate – e.g. irregular data
  - Overhead too large – performed in software
- Memory bound
  - Memory bandwidth strangled
  - Memory access latency isolation – 100’s, 1000’s cycles
  - 100X time to touch all of memory
  - Memory hierarchy dies with low temporal locality
  - TLB coherency and scalability
- Too difficult to program – resource management by hand
- Too big, too hot, too $$
- Vulnerable to reliability failures
Multi Threaded Vector with MIND
Locality Driven Architecture

- Distributed shared memory
  - Not cache coherent
  - Support for atomic distributed object operations
- High efficiency through dynamic management of latency and overhead
  - Will make programming easier by simplifying burden of resource control
- Two execution regimes:
  - High temporal locality
  - Low or no data reuse
- Very high-speed Multithreaded Vector compute processors (MTV)
  - Multithreaded vector architecture
  - Compile time managed caches with hardware assist
  -Parcel message driven computation and synchronization
- Processor in Memory data processors (MIND)
  - Expose very high memory bandwidth while reducing latency/power
  - Perform low/no reuse data ops directly in memory
  - Support system overhead functions for high efficiency
- *Percolation* prestaging for initializing computations to hide latency and overhead as well as providing dynamic load balancing
Next Generation HPC Architecture
with Dynamic Adaptive Resource Management

- MTV 0
  High-speed Buffer

- MTV 1
  High-speed Buffer

- MTV N-1
  High-speed Buffer

- Low Latency Network

- Global Shared High-speed Buffer

- High Bandwidth Optical Network

- MIND 0
  3/2 RAM

- MIND 1
  3/2 RAM

- MIND P-1
  3/2 RAM

Dynamic Scheduling, Load Balancing
Percolation (data/task prestaging) Parcels

Data Intensive User Function Instantiation

June 24, 2002
Thomas Sterling - Caltech & JPL
Current PIM Projects

- IBM Blue Gene
  - Pflops computer for protein folding
- Stanford Streaming
  - Power efficient
- UC Berkeley IRAM
  - Attached to conventional servers for multi-media
- USC ISI DIVA
  - Irregular data structure manipulation
- U of Notre Dame PIM-lite
  - Multithreaded
- Caltech MIND
  - Virtual everything for scalable fault tolerant general purpose
Why is PIM Inevitable?

- Separation between memory and logic artificial
  - von Neumann bottleneck
  - Imposed by technology limitations
  - Not a desirable property of computer architecture
- Technology now brings down barrier
  - We didn’t do it because we couldn’t do it
  - We can do it so we will do it
- What to do with a billion transistors
  - Complexity can not be extended indefinitely
  - Synthesis of simple elements through replication
  - Means to fault tolerance, lower power
- Normalize memory touch time through scaled bandwidth with capacity
  - Without it, takes ever longer to look at each memory block
- Will be mass market commodity commercial market
  - Drivers outside of HPC thrust
  - Cousin to embedded computing
Limitations of Current PIM Architectures

- No global address space
- No virtual to physical address translation
  - DIVA recognizes pointers for irregular data handling
- Do not exploit full potential memory bandwidth
  - Most use full row buffer
  - Blue Gene/Cyclops has 32 nodes
- No memory to memory process invocation
  - PIM-lite & DIVA use *parcels* for method driven computation
- No low overhead context switching
  - BG/C and PIM-lite have some support for multithreading
Attributes of MIND PIM Architecture

- **Memory, Intelligence, and Network Device**
  - Parcel active message driven computing
    - Decoupled split-transaction execution
    - System wide latency hiding
    - Move work to data instead of data to work
  - Multithreaded control
    - Unified dynamic mechanism for resource management
    - Latency hiding
    - Real time response
  - Virtual to physical address translation in memory
    - Global distributed shared memory thru distributed directory table
    - Dynamic page migration
    - Wide registers serve as context sensitive TLB
  - Graceful degradation for Fault tolerance
    - Graceful degradation through highly replicated fine-grain elements.
    - Hardware architecture includes fault detection mechanisms.
    - Software tags for bit-checking at hardware speeds; includes constant memory scrubbing.
    - Virtual data and tasks permits rapid reconfiguration without software regeneration.
A Software Architecture

- Conventional Programming Models
- Advanced Programming Models
- Global Runtime System
- Local Runtime System
- Operating System
- I/O
- Mass Storage
- Hardware Architecture

compilers
MIND Programming Opportunities

- Provides lightweight efficient threads
- Reduces overhead burden
- Avoids data access latency
- Rapid streaming for associative searches
- Exposes massive parallelism
- Enables efficient and concurrent gather-scatter
- Enables efficient and concurrent array transpose
- Permits fine grain manipulation of sparse and irregular data structures
MTV/MIND Programming Challenges

- Decoupled activities
  - Compiler/programmer must delineate between two classes of work
- Heterogeneous
  - Two or more separate instruction sets
  - More than one uniform cost function
- Coherence among cache and memory copies
- Separate ALU, IP, and Memory
  - Not conventional node, semi-independent and self-contained
  - Part of activity state, shared with other nodes
- Portability of legacy codes
  - E.g. Fortran, C, MPI
Never Predict a new language
There will be a new language
  - Called BOCCE from C3PO New Republic Software Inc.; $49.95, delivery 2007
Expose diversity of parallelism
Reflect a semantics of affinity
  - Support active decision space for allocation and scheduling
Provide easy code structuring for partial rollback & restarts
  - Micro-checkpointing for fault tolerance
Integrate performance information as part of regular output
  - Possibly with intelligent closed-loop
Exploit dynamic adaptive resource management mechanisms
  - Multithreading, percolation, parcels
Keeping MPI in MIND

- MPI as programming model for MTV/MIND
  - Migration of legacy codes: MPI with C or Fortran
  - Transition methodology for leveraging learning-curve
    - Reduce barriers to getting new applications from old programmers
  - An excellent programming model for certain classes of parallel algorithms
- Use as intra-MPI process accelerator or as MPI process host
  - Unlike most accelerators, data already in MIND PIM chips
- Incremental performance enhancements
  - Runs normally on first startup
  - Target data intensive performance bottlenecks
  - Build on libraries with conventional interfaces and new wrappers
- Spanning multiple MIND nodes with MPI process
- Parcels perform MPI message requirements and also migrate work
- Collective operations
  - Very efficient MPI implementations with MIND/PIM
- Threaded MPI processes
  - Exploit MPI process internal parallelism
Percolation of Active Tasks from MIND to MTV

- **Latency**
  - Prestage all relevant data near execution engine
  - Multiple stage latency management methodology
  - Augmented multithreaded resource scheduling

- **Heterogeneity**
  - Compute intensive work in high speed processors
  - Low data-reuse work in memory processors
  - Hierarchy of task contexts

- **Overhead**
  - Perform all data manipulation and scheduling in separate logic
  - Coarse-grain contexts coordinate in PIM memory
  - Ready contexts migrate to SRAM under PIM control releasing threads for scheduling
  - Threads pushed into SRAM/CRAM frame buffers
  - Task stack loaded in register banks on space available basis
  - Write back to main memory
Panel Questions and Issues

- Which are the key requirements of HPC applications?
  - Support for substantial and diverse abstract parallelism
  - Locality management through a nomenclature of affinity
  - Gradual performance sensitivity to adjustments in application code and system resource parameters and availability
  - Convenient restart points for transparent resumption in the event of a fault

- What is wrong with the SOA in programming HPC systems?
  - Nothing, if you are a sadomasochist
  - There is a close and uncompromising relationship between the programmer and the physical properties of the HPC machine structure.
  - Limited in the classes and forms of parallelism that can be exposed
  - Precludes exploitation of runtime information

- Which major directions will HPC architecture development take? Will they make programming simpler or even more difficult than today?
  - Plan A: The lowest common denominator – clusters of commodity SMPs (harder)
  - Plan B: The right thing – MTV with MIND altering products (simpler)
Panel Questions and Issues (2)

How will High-level languages, compilers, runtime systems and software tools evolve during this decade? Which new features are required to support fault tolerance?
- Impossible to predict
- Emergence of runtime agents enabled by cheap pervasive logic
- Compiler to runtime relationship through intermediate API
- Active decision choice space
- Introspection or monitor threads for resource measuring and fault detection

How can legacy codes be efficiently ported to new architectures?
- At the risk of being Clintonesque, how do you define “efficiency”
  - Definition A: as good as it would run on conventional system of comparable capacity
  - Definition B: as good as it should run with superior properties of new system
  - Yes to A, No to B
- Incremental rewrites required for adaption of applications to new machines
  - O/S services
  - Common shared libraries
  - User critical sections