Parallel Computing: A View From Berkeley

E. M. Hielscher

6 February 2008
Motivation

- All major processor manufacturers have switched to parallel architectures
- This switch driven by three “Walls”: the Power Wall, Memory Wall, and ILP Wall
- Power = Capacitance × Voltage^2 × Frequency
- Thus frequency scaling cannot continue forever, as increases in frequency demand more power - a Pentium 4 is already as hot as a hot plate
- The gap between memory access speed and processor speed increases by 50% per year
- ILP techniques such as OoO execution and VLIW have diminishing returns (see e.g. Alpha 21464)
Motivation

- Result: Moore’s Law continues, but only in terms of the number of transistors per given area. We’re no longer seeing speedups we’re use to:
Big Change

- Parallel processing is nothing new
- In the past, research in PP was driven by new breakthroughs which opened design space, with uniprocessors always prevailing in the end
- Now, there are immense difficulties in the uniprocessor realm which make parallel processing less difficult - hence the big push toward parallelism
Recommendations

- Overarching goal should be to make it easy to write programs which execute efficiently on highly parallel computing systems
- Target should be 1000s of cores per chip, as these are most efficient in MIPS/W, MIPS/cm², MIPS/$
Conventional Wisdoms

1. Old CW: Power is free, but transistors are expensive
   1. New CW: We can put more transistors on a chip than we have power to turn on
2. Old CW: Dynamic power is the only power concern
   2. New CW: For desktops and servers, static power leakage can be 40% of total
3. Old CW: Monolithic processors are reliable internally, with errors only at the pins
   3. New CW: As chips drop below 65nm feature sizes, they will have high error rates
Conventional Wisdoms

4. Old CW: Building upon the past, we can continue to raise level of abstraction and the size of hardware designs

4. New CW: Wire delay, noise, cross coupling, manufacturer variability, reliability, validation, etc. increase cost of large designs below 65nm

5. Old CW: New architecture ideas are demonstrated by building chips

5. New CW: Cost of masks at 65nm, ECAD software for designing chips, and design for GHz clock rates mean that building believable prototypes no longer feasible
Conventional Wisdsoms

6. Old CW: Performance yields both lower latency and higher bandwidth
6. New CW: Bandwidth now often improves with square of latency
7. Old CW: Multiply is slow, but load and store is fast
7. New CW: Memory Wall: load and store slow, mul fast
8. Old CW: We can get more ILP from compiler and architecture innovation (e.g. OoO, Speculation, VLIW)
8. New CW: Diminishing returns to finding more ILP
Conventional Wisdoms

9. Old CW: Don’t bother parallelizing your application, as you can wait a little while for a faster sequential computer

9. New CW: A faster sequential computer won’t arrive for a long time

10. Old CW: Uniprocessor performance doubles every 18 months

10. New CW: Power Wall + Memory Wall + ILP Wall = Brick Wall. Uniprocessor performance now only doubles roughly every five years
ConventionalWisdoms

11. Old CW: Increasing clock frequency is the primary method for improving processor performance
11. New CW: Increasing parallelism is the primary method for improving processor performance
12. Old CW: Less than linear scaling for a multiprocessing application is a failure
12. New CW: Any speedup via parallelism is a success
Multicore Not the Solution

- Multicore refers to the doubling of the number of cores on chip with every chip generation
- Switching from sequential programming to moderately parallel programming brings with it great difficulty without much improvement in power-performance
- Thus this paper’s position is that we desperately need a new solution for parallel hardware and software
- The vision is manycore, thousands of cores on chip, and new architectures and programming models are welcomed if they simplify the efficient programming of such highly parallel systems
Positive Changes

- Moore’s Law continues, so we can soon realize economical chips with 1000s of cores
- Monolithic manycore chips can have very low latency and very high bandwidth, open up design space in architecture and programming models
- FOSS allows software stack to evolve much more quickly
Seven Questions

Applications
1. What are the applications?
2. What are common kernels of the applications?

Programming Models
5. How to describe applications and kernels?
6. How to program the hardware?

Hardware
3. What are the hardware building blocks?
4. How to connect them?

Evaluation:
7. How to measure success?

Figure 1. A view from Berkeley: seven critical questions for 21st Century parallel computing. (This figure is inspired by a view of the Golden Gate Bridge from Berkeley.)
Applications and Dwarfs

- Traditional method of evaluation of architecture is through a benchmark suite based on existing programs
- It is currently unclear the best way to express a parallel computation
- Thus using existing code to drive parallel computing investigations is unwise
- Higher level of abstraction needed: dwarfs
Dwarfs

- Capture a pattern of computation and communication while avoiding details about the underlying architecture
- Original seven come from high performance computing; as long as no more than a few dozen are compiled, more should be added to cover additional areas
- Studied Machine Learning, Databases, and Computer Graphics and Games to explore the need for more dwarfs
Original Seven Dwarfs

- Dense Linear Algebra - Dense datasets
- Sparse Linear Algebra - Data w/ many zeros
- Spectral Methods - e.g. FFT
- N-Body Methods - interactions between many discrete points, e.g. Barnes-Hut
- Structured Grids - Regular grid; points are conceptually updated together, e.g. PDE solvers
- Unstructured Grids - Irregular grid often with many levels of indirect memory access, e.g. vector computers with gather/scatter
- MapReduce - Parallel computation on subsets of data, combined in an end-step
Added Dwarfs

- Combinational Logic - Simple operations on massive amounts of data (e.g. encryption)
- Graph Traversal - Indirect lookups in search; little computation (e.g. Quicksort)
- Dynamic Programming - Problem solving via solving simpler sub-problems
- Backtrack and Branch-and-Bound - Global optimization for huge search spaces
- Graphical Models - Bayesian Networks and Hidden Markov Models
- Finite State Machines - Interconnected States as used in parsing; embarrassingly sequential
Dwarf Summary

- Intel is also investigating the categorization of computing workloads in a similar fashion
- The 13 dwarfs are likely not exhaustive, but a chart is given showing that they do cover a vast array of computational problems
- Another issue involves how to compose dwarfs to solve problems which span multiple dwarfs
An inflection point has been hit in processor design: bigger is no longer better due to factors such as power consumption and inability to efficiently design complex modules.

What is the best building block for designing processors?
Advantages of Small Processor Building Blocks

- Parallelism is an energy-efficient way to achieve performance
- Many small cores give highest performance per unit area for parallel code
- Larger number of smaller cores allows finer grained voltage and frequency scaling
- Small cores are easy to shut down in the face of failure or manufacturing variation
- Smaller core is easier to design and formally verify
- Smaller hardware modules are more power efficient and their power consumption patterns easier to predict
Optimal Processing Element

- Traditional studies have dependencies on uniprocessor benchmark suites, process technology, and workloads and thus are difficult to generalize to the parallel case
- However, simple pipelining was shown to be beneficial to energy-delay product, while superscalar features were not worth their hardware
- Belief that the efficient building block of future architectures will be:
  - Simple
  - Modestly pipelined (5-9 stages)
  - Consist of Floating Point Units, Vector Units, and SIMD elements
Memory

- Memory Wall is the main obstacle to good performance in half the dwarfs
- Should explore innovation in memory designs to increase memory bandwidth
- Further, hardware cost is increasingly shifting to memory from processing
• Traditionally crossbar switches were used to connect processors to cache banks in multiprocessors, but as the silicon required scales with the square of the number of processors, this isn’t feasible in manycore designs

• Research must be done on communication patterns in parallel computations using the dwarfs

• Research must be done on how to efficiently provide for both bandwidth-oriented and latency-oriented communication, as both are important
Communication Primitives

- Inter-core bandwidth in a CMP can be an order of magnitude greater than that in an SMP
- Inter-core latencies in a CMP are at least an order of magnitude less than in an SMP
- CMPs could offer light-weight coherence and synchronization primitives for use on chip
- Much architecture and algorithm design space to be explored
Synchronization

• Locking is notoriously difficult to program for many reasons (deadlock, non-composability)
• Locking is also wasteful in that it busy-waits or uses interrupts/context switches
• Transactional Memory is a promising but active research field which looks to provide an easier to use primitive
• Other potential primitives include full-empty bits and message passing
Performance Counters

- Performance and power consumption are now the responsibility of the compiler writer and systems programmer
- Thus performance counters are necessary to provide information on any feature that significantly affects performance or power consumption
Programming Models

- A programming model serves as a bridge between a developer’s natural conception of an application and an implementation on given hardware
- Two conflicting goals for programmer: productivity and implementation efficiency
- Abstraction of underlying hardware details allows for increased productivity
- Visibility of underlying hardware details allows for specific performance tuning and better efficiency
A key factor of manycore designs’ success will be the ability of programmers to harvest their performance potential.

In fact, developing programming models that productively enable development of highly efficient implementations of parallel applications is the biggest challenge facing the deployment of future manycore systems.
Recommended Characteristics

- Use of results from psychology in design
- Make programs independent of the number of processors
- Support a rich set of data types
- Support styles of parallelism that have proven successful in the past
Compilers vs. Autotuners

- Traditional compilers have gotten very large and complex, so modifying them to produce automatically parallel code seems like a hard problem.
- Autotuners are a promising solution, which try many configurations of a library on a platform during installation to provide an automatically tuned version.
Deconstructing OSes

- Authors believe that virtual machines should provide the ability for applications to select only those features of the OS they need to operate
- Embedded computers increasingly align with high performance computation in their need for protection
- New ISAs should directly support virtualization
• Should develop metrics for qualitative comparison of programmer productivity using different models
• Metrics for evaluating performance should focus on:
  • Minimization of remote accesses
  • Load balancing
  • Large granularity of data movement and synchronization
• RAMP project looks to provide a way for researchers to build realistic processor prototypes using FPGAs