ON ELIMINATING THE INTERPRETATION LAYER IN FUTURE EXTREME-SCALE CLOUD CENTER ARCHITECTURES

Dr. Kemal Ebcioglu
President, Global Supercomputing Corporation
December 10, 2013
IEEE/ACM MICRO-46 Symposium
UC Davis, California, USA

Talk outline

- Short retrospective on IBM Research VLIW project
- Our future vision for cloud data centers
  - First pain point: energy inefficiency
  - Second pain point: lack of performance with high productivity
  - Third pain point: inefficiency of resource sharing
- A compiler demo:
  - Converting sequential code to an application-specific supercomputer
- Future uses of our cloud data center technology

A short retrospective on VLIW, binary translation and Java performance work at IBM Research

- IBM Research VLIW project
  - Launched in 1986. Focused on improving the performance of branch-intensive, non-numerical code. Included:
    - A hardware prototype, object code translation, three generations of compilers,“scalable” RISC-like VLIW instructions, use of VLIW compiler techniques for existing superscalars
  - IBM Research VLIW compiler techniques can:
    - Schedule operations on multiple paths as soon as their operands are ready, but stop executing the remaining operations on a given path as soon as it is known that that path will not be taken.
    - Recognize common computations on multiple paths and execute them only once.
    - Achieve variable iteration initiation intervals during software pipelining of loops with conditional branches
  - Found solutions for:
    - Precise exceptions in the presence of speculative execution
    - Memory mapped I/O in the presence of speculative execution
    - Multiprocessor consistency
    - Object code translation of self-modifying code

A short retrospective on VLIW, binary translation and Java performance work at IBM Research (continued)

- Over 30 project contributors throughout the years, including highly respected colleagues
- DAISY dynamic binary translation and optimization project
  - 100% compatibility with original PowerPC code (including OS)
  - Liberates hardware designer from having to implement ISA verbatim
  - Allows architectural convergence (single HW implementing multiple ISAs)
- LaTTe (with Seoul National U.): Java compilation techniques for
  - fast register allocation
  - thread synchronization
  - exception handling
  - just-in-time optimizations
  - just-in-time instruction scheduling
  - garbage collection

One retrospective observation on the IBM Research VLIW project

- What we achieved:
  - Near optimal parallel performance within an entire loop invocation of arbitrary branch intensive non-numerical code
  - No penalty due to branch mispredictions, thanks to new multi-way branching hardware
  - Without requiring predication (when control dependence is on a dependence cycle, predication may slow execution down)
- What we missed:
  - Serialization was incurred on every inner or outer loop entry and exit
- Avoiding the serialization would have opened floodgates of parallelism (hard problem)

Our future vision: Some deficiencies of current computer architecture research

- Architecture research has stayed within the **processor box**
  - Past: CISC vs. RISC, in-order vs. out-of-order superscalar
  - Present: multi-cores, graphics processors
  - But we have just been putting more of the processors we know into a chip. We need a more innovative use of the extra area!
- Ingrained engineering culture is to design a processor and **throw the architecture design over the wall**
  - ISA is a hardware-software contract
  - Hardware team asking the software team to make applications run better/more parallel
- But a sequential software application and its highly parallel, application-specific supercomputer hardware implementation are one and the same, in this sense:
  - the former can be automatically converted to the latter
- The **wall** between hardware and software **does not and need not exist!**

Our future vision: On ***interpretation*** in computer architecture

- Difference between software interpreters (e.g., JVM) and compiled code is known, but interpreters are not confined to software
- Interpreters are pervasive today in computer architecture
  - A processor is a parallel hardware implementation of an *instruction set interpreter*
  - Maurice Wilkes invented microprogramming, or a *state machine interpreter* (gave rise to VLIW with compiler innovations of Josh Fisher and others)
  - An FPGA is an *interpreter of arbitrary digital circuits*
- What is not an interpreter:
  - An application-specific multi-chip parallel ASIC supercomputer implementation of a given software algorithm is not an interpreter.
  - Multi-chip parallel ASIC supercomputer is the best (lowest power, highest performance) “native” implementation of an algorithm.

Our future vision: eliminate interpretation for high parallelism and low power, but increase ease of use and enable resource sharing for cloud computing

- Remove the level of ISA interpretation
  - Except for backward compatibility and low reuse apps
- Automatically convert a sequential software application directly to an application-specific multi-chip ASIC supercomputer
  - For high programmer productivity
- But share hardware resources for implementing a cloud data center
  1. An ASIC executes one function only
  2. Cloud computing uses general purpose processors which can run any application
- Exploit the unexplored space in between these two alternatives (e.g., compiler generated reconfigurable ASICs), to retain the ASIC power and performance advantages.

A supercomputer example

***TİANHE-1A* Supercomputer (2010)**
2,5 petaflops
14336 Intel Xeon processors
7168 NVIDIA Tesla graphics processors

The first pain point in today's data centers: Energy Inefficiency

- Today's general purpose processors are reaching an innovation plateau, in terms of performance and energy efficiency
  - Today's general purpose processors are encountering (1) **frequency**, (2) **power**, (3) **design complexity**, and (4) ***memory wall*** barriers
  - **Design complexity** pushes innovation away:
    - A simple change idea often does not perform well and is discarded
    - A complex change idea may perform well but (if incorporated) makes the design more complex and more resilent to innovation
  - **Energy is continually being wasted** on the interpretation of an instruction set architecture

An energy efficiency vision

- We believe energy efficiency should be achieved not in the interpretation of the instruction set architecture, but within the software program's algorithm itself
- Our objective is to change traditional computer architecture to make applications more energy efficient
  - Application-specific, low power designs
  - Systematic optimizations for reducing power
    - Application specific memory hierarchy
    - Application specific techniques for reducing communication distance
    - Application specific techniques for reducing communication volume
- Some target applications
  - Mobile platforms
  - Satellite systems with limited power
  - Exascale cloud data centers
    - Google's data centers require 300 Mega Watts of power
      http://www.nytimes.com/2012/09/23/technology/data-centers-waste-vast-amounts-of-energy-belying-industry-image.html

Second pain point: The difficulty of writing parallel programs and the limited performance of general purpose processors

- Multi-core processors and graphics processors perhaps promise lower power and higher performance;
- However:
  - These technologies reduce the ***productivity of programmers*** who are used to the standard processor model.
  - Writing parallel programs is very difficult for users.
  - There is not quite a solution within the Industry
    - Intel, Tim Mattson, *parallel programming patterns*
    - Cray, Cascade; IBM, X10; experimental parallel programming languages
    - Historical parallelization techniques DoAcross, DoPipe still not efficient with existing processor synchronization overheads

A vision for accelerating computations

- Developing a **compiler for automatically parallelizing single threaded programs**
  - The solution was not where one would expect it to be (multi-core architectures + parallelizing compiler)
  - Compiler generated, application-specific synchronization hardware is one key to the solution (see our patent apps)
- Design and implementation of application-specific supercomputers
  - C/C++ → scalable IP core to create supercomputer
    - Register transfer level
  - Low power
  - High performance
  - Compiling in hardware reliability with flip-flop level redundancy

Third pain point: inefficient resource sharing

- Purely application-specific hardware solutions are not always sufficient. Sharing hardware resources is also sometimes required.
- Today resource sharing is achieved in the best way with **service provisioning via cloud computing**, which brings forth economies of scale, and has become one of the most successful and important IT trends of our age.
- Cloud computing has become a *disruptive technology*. The market size for cloud computing is expected to increase to **241 billion dollars** in 2020. http://finance.yahoo.com/news/american-diversified-holdings-corporation-adhc-142400999.html

Third pain point (continued)

- However, cloud computing is done today with general purpose processors and normal software, and cloud systems are obligated to spend unnecessary energy repeatedly.
- Today's Hypervisors and Operating Systems are not designed in a scalable manner for tomorrow's cloud systems with extreme scale parallelism, and the hardware is not resilient.
- In today's hardware, general purpose processors and perhaps FPGAs are used. But both are slow and energy inefficient as compared to ASIC implementations.

An efficient resource sharing vision

- Our objective is low power, high performance, scalable and reliable hardware-accelerated cloud computing.
- Resource sharing using semi-reconfigurable ASICs
  - Special purpose, low power ASIC modules that can take on multiple functions according to configuration parameters.
- Implementing a scalable, parallel hypervisor algorithm for realizing resource sharing completely in hardware
  - Freedom from power consuming processor instructions and hypervisor software, which adds extra load onto the system
  - Supporting *Service Level Agreements*

Published patent applications by Global Supercomputing Corporation- 1

- “Parallel hardware hypervisor for virtualizing application-specific supercomputers”
- Co-inventors: Dr. Kemal Ebcioglu, Dr. Atakan Dogan, Reha Oguz Altug, Dr. Mikko Herman Lipasti, Eray Ozkural.
- 103 page specification, 15 drawings, 17 claims. Filed on 02/04/2012 as US non-provisional patent application 13/366,318. Describes a new low-power all-hardware cloud computing system based on application specific virtual supercomputers.
- http://patentscope.wipo.int/search/en/WO2013115928

Published patent applications by Global Supercomputing - 2

- “Method and system for converting a single-threaded program into an application-specific supercomputer”
- Co-inventors: Dr. Kemal Ebcioglu, Emre Kultursay, Dr. Mahmut Taylan Kandemir.
- 223 page specification, 70 drawings, 35 claims. Filed on 11/15/2011 as US non-provisional patent application 13/296,232.
- Describes new methods that create low-power, multi-threaded, multi-chip supercomputer hardware automatically starting from single threaded software.
- http://patentscope.wipo.int/search/en/WO2013074341

Published patent applications by Global Supercomputing - 3

- “Storage unsharing”
- Co-inventors: Emre Kultursay, Dr. Kemal Ebcioglu, Dr. Mahmut Taylan Kandemir.
- 55 page specification, 8 drawings, 23 claims. Filed on 6/9/2011 as US non-provisional patent application 13/156,881.
- Describes new methods for partitioning the memory image of a software program into independent, hierarchical, and application-specific smaller memory units, for the purpose of bringing data closer to the computation units, energy efficiency and performance.
- http://www.google.com/patents/US20110307663

A compiler demo

- RandomAccess benchmark
- Sha-1 hashing algorithm

Future uses of application-specific virtual supercomputers

- Pervasive use of application-specific virtual supercomputers for provisioning of services in cloud centers, e.g.:
  - Health care (bioinformatics),
  - Internet search
  - Critical time consuming EDA tools for VLSI design
  - Financial applications of banks
  - Data mining
  - BitCoin cloud system
  - Graph search algorithms (social network backbone)
  - Artificial Intelligence decision making algorithms (alpha-beta pruning)
  - Video surveillance
  - Satellites with limited power environments
  - Many others …

Future uses of application-specific virtual supercomputers (last slide)

- Artificial Intelligence
  - Application specific virtual supercomputers will overcome performance barriers for AI, enabling search and learning algorithms to do more.
  - We believe: Successful AI programs will likely be based on the introspection of a researcher herself/himself or on the analysis of the introspection of others
  - E.g., musical creativity of computer algorithms (our CHORAL project)
    - Computational power required
    - Many absolute rules to ensure quality required
    - Many heuristics to guide combinatorial search required
  - ***Emotional content not a stumbling block*** of computer-generated music
  - Question: some mental discoveries (e.g., a difficult proof, a beautiful compositional idea) "arrive by sudden inspiration"
    - cannot be analyzed by introspection
  - Can the knowledge base of rules and heuristics plus combinatorial search ever meet or exceed the performance of its programmer/composer team?
    - Including the behavior not explainable by introspection