EXISTING APPLICATIONS OF BOUNDARY MATH FOR MULTIPROCESSING

William Bricken
April 2004

SUMMARY

Boundary Math has been applied to software and hardware optimization, to formal verification, to fine and course-grain parallel processing, to various improvements in the software utilization of parallel architectures, to reconfigurable hardware design, and to EDA logic synthesis tools. All applications have been in the context of commercial research.

Over the last two decades I've applied the Boundary Math techniques to a range of computational problems, with the last several years concentrated on area and delay minimization of circuit netlists.

All of my work in Boundary Math has been for proof-of-concept, as demonstration of potential rather than as direct commercial application. Several potentially relevant examples follow below. Their sequence of presentation reflects the temporal evolution of Boundary Math capabilities.

There is a rigor that bounds these examples; in all cases, the work:

-- is innovative and first-of-its-kind.
-- is implemented as verifiable computer programs or simulations.
-- has direct application relevance and is not abstract mathematics.
-- has been developed in commercial (not academic) research settings.
-- has been reviewed by corporate and/or public groups.
-- has available technical reports and implementation code

Thus, the information below describes substantive capabilities and benefits that are exemplified by the selected application areas and problems addressed. None have been empirically demonstrated in commercial products. Naturally, the specific commercial benefits for each Boundary Math commercial research application are highly sensitive to the technical and marketing context they are embedded into.

Applications briefly described herein include:

Software Verification, Optimization and Decomposition
Theorem-Proving and Verification Engines
Fine-Grain Multiprocessor Parallelism
Course-Grain Multiprocessor Parallelism
Various Multiprocessor Capabilities
Reconfigurable Hardware
Circuit Design Optimization
Numerical Co-Processor

SOFTWARE VERIFICATION, OPTIMIZATION and DECOMPOSITION

This project was a component of a larger program to design a programming language that was responsive to semantic as well as syntactic editing. We modeled the data and control flow within Ada programs, using Boundary Math to provide flow models of computation, to optimize these models, and to identify parallel and sequential components. The deliverable was a software program that successfully

- decomposed Ada programs into fine-grain components that could be edited and executed in parallel,

- identified necessary sequential paths that placed a lower bound on execution speed,

- verified the correctness of the internal model and optimized the original Ada code,

- identified difficult to detect programming faults and semantic structures such as

  -- invocations that necessarily followed a given process at a later time.
  -- invocations that used specific variables and function calls, even when those forms were not within a syntactic scope.
  -- forms that changed value or structure within a given scope.
  -- variables that were used prior to being assigned a value.
  -- functions that are invoked only when conditionally needed.

These tools can be applied to the compilation (refinement, verification and optimization) of parallel programming languages and to resource allocation in a distributed hardware environment.

THEOREM-PROVING and VERIFICATION ENGINES

A theorem-prover can be used for verification of hardware design and of software programs. We implemented and rigorously benchmarked a Boundary Math theorem-prover against widely used formal verification tools such as the Boyer-Moore Theorem Prover, showing superior performance efficiency.

We then implemented the Boundary Math tool as a parallel logic engine that separated parallel and sequential aspects of formal proof. This engine simulated parallel deduction on a sequential machine.

In another aspect of this project, we extended the Boundary Math theorem prover to operate in the presence of contradictory data. Thus in the presence of a contradiction (which may come for different data bases, from

conflicted specifications, or from a design error), the theorem prover computed as much as was possible without implicating the specific set of contradictions.

These tools can be applied to verification of distributed computation, to control partitioning and load balancing across processors, and to fault tolerant computing.

FINE-GRAIN MULTIPROCESSOR PARALLELISM

We successfully implemented the above parallel logic engine on a 16-node Intel Hypercube, demonstrating hardware parallelism. Although the Hypercube supported only course-grain parallelism, the Boundary Math model supported parallel decomposition at any grain-size, from single computational steps to work sufficient to occupy a CPU. No control nodes or dispatchers were used, instead a computation was initially distributed over the available nodes, and then the result emerged solely from local, asynchronous, parallel communication between these nodes.

The system was used to make deductions from a distributed AI rule-base, and to simulate evaluation of circuit fragments. This result demonstrated that the Boundary Math techniques could be applied comprehensively, from decomposition of serial programming into parallel components through distribution of and computation with these components on parallel hardware. The capability can be used for many aspects of software programming and efficiency optimization of concurrent processors.

COURSE-GRAIN MULTIPROCESSOR PARALLELISM

We used Boundary Math techniques to design the communication structure of a virtual reality system. We built an operating system shell within a distributed UNIX environment that coordinated the rigorous demands of real-time immersive virtual environments. This included real-time integration of several sensor measurements of a participant's location, movement, and command instructions; computation of the dynamic 3D graphics environment with physical modeling and many independent active software agents; and display at 16 frames per second of the multiple participants, the virtual environment and the interaction between the two.

The VEOS project demonstrated that Boundary Math can be used to design, organize and structure high-end, real-time, interactive computational environments that incorporate sensor fusion, dozens of distributed processors, and real-time graphics display.

These techniques confirm the generic utility of Boundary Math for course-grain distributed processing under severe timing constraints.

VARIOUS MULTIPROCESSOR CAPABILITIES

We applied Boundary Math to address several different issues in real-time parallel processing. These capabilities were all simulated in software, and not translated to operational hardware. All techniques were applied both to software programs and to circuit designs, although to differing degrees.

- SUPER-RISC ARCHITECTURE

We developed a generic set of assembly-level instructions for a Boundary Math microprocessor. The processor architecture is conventional, only the machine instructions change. The instruction set consists of five instructions: two basic instructions for computation, two for reading and writing into memory, and one to terminate. Note that no ALU is required, since arithmetic, logical, shift, compare, and move instructions are not necessary. The Boundary Math processor is essentially a hardware emulator.

Due to the context-free nature of the Boundary Math instruction encoding, straight-line programs (without loops and branches) can be used when desirable. Like hardware, branching instructions are incorporated using multiple reference rather than multiple entry. A straight-line processor computes using two JUMP instructions, skipping over other instructions that are unnecessary for evaluation within the current binding regime. Unbound variables can passed unevaluated, providing a partial evaluation engine.

- BIT-STREAM CIRCUIT SIMULATION

We converted Boundary Math representations of both software (programming constructs) and hardware (circuit designs) into bit-streams and then implemented in software a single-pass, non-backtracking processor that evaluated these bit-streams. The bit-stream representation was used to measure the inherent sequential complexity of the object code, to provide an efficiency comparison between different design structures and styles, and to place upper and lower bounds on design performance given any input.

We designed but did not simulate a hardware architecture for the bit-stream processor that required only a couple of hundred logic gates. We applied parallel decomposition to the bit-streams to compute the potential efficiency of massively parallel bit-stream processors.

We also applied Boundary Math optimization to the bit-streams and concluded that for real-time bit-streams it is cheaper to evaluate non-optimal forms than to optimize and then evaluate. Several details of the hardware implementation are not completed.

The parallel engine could theoretically provide very fast through-put at very high dataflow rates. It could be used in combination with various compiler techniques to facilitate remote evaluation and late binding.

- MODULE IDENTIFICATION and ABSTRACTION

We designed and partially implemented a set of tools for partitioning, placing, and routing computational structures into a hardware environment. The tools included Boundary Math structural transformations that emphasized each particular type of available abstraction. Thus the output of these tools could be steered toward desired architectural characteristics such as less logic, less routing, and/or fewer modules.

The abstraction tools provided the following capabilities:

-- Functional Abstraction: automated identification and abstraction of repeated design components, especially when the components were not specified in the design specification.

-- Bit-width Abstraction: automated generation of arbitrary bit-width computational components.

-- Vector/Matrix Abstraction: automated identification and abstraction of data vectors and matrices of any degree. The Boundary Math optimization techniques are applied using matrices as elementary objects.

These tools were used for FPGA optimization, and could be generalized to multiprocessor optimization. Steerable partitioning can help with architectural load and routing balancing, and with efficient placement of processes within a processor array.

RECONFIGURABLE HARDWARE

We designed and simulated in SPICE a Boundary Math reconfigurable hardware platform (CoMesh) with design trade-offs guided by Boundary Math algorithms. Software placement and routing tools for the reconfigurable logic closely map netlist specifications onto the hardware architecture.

The CoMesh design outperformed in simulation Xilinx Virtex parts, providing three times the logic density, while running at a stable speed of 300 MHz. This design is intended to be competitive in the FPGA marketplace.

CIRCUIT DESIGN OPTIMIZATION

We designed and implemented an EDA logic synthesis and mapping tool that incorporated much of the prior work in Boundary Math. The Iconic Logic Optimizing Compiler (ILOC) has been demonstrated to outperform the Synopsys logic synthesis tools in both logic reduction and delay reduction for a diversity of circuits.

Logic synthesis of large hardware designs provides the most rigorous test of Boundary Math optimization techniques. The current ILOC alpha/prototype system has optimized several ALU architectures and hundreds of design components up to 20,000 gates. The current software has a programmer's interface and is not ready for design engineers.

Netlist optimization can be generalized to transform structural models based on measurable structural characteristics, for any system of relatively homogeneous nodes with semi-structured local transactions.

NUMERICAL CO-PROCESSOR

Boundary Math can be interpreted as and used for floating-point binary arithmetic. Boundary-based numbers are currently implemented in software; the hardware versions are rough designs ideas. This potential application provides new computational models for hardware design by mapping arithmetic directly onto non-logical hardware cells. The new models may have very desirable implementation and performance characteristics.

CONCLUSION

The capabilities of Boundary Math optimization techniques have been thoroughly explored in software programs and simulations. They range over much of the hardware and software design territory for reconfigurable multiprocessor arrays. Recent software development has been tightly focused on competitive ASIC logic synthesis for design area and delay. The Boundary Math tools have been rigorously tested, but not within the context of commercial applications and problems. They also lack many of the fine-grain special cases necessary for commercial software. The Boundary Math capabilities have evolved through proof-of-concept and have not seen the investment of commercial refinement.