Computing, Approximately

Ravi Nair and Daniel A. Prener
IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598

Computation today brings with it an expectation of preciseness – preciseness in the definition of the architecture, preciseness in the implementation of the architecture, and preciseness in the program designed to solve problems of interest. But is such preciseness important when the program itself encodes an approximate solution to a problem and is not sacrosanct? Is such preciseness important when an instruction executed by a program does not need all the restrictions indicated by its definition in the architecture, and hence does not make use of all the hardware associated with executing the instruction? Is such preciseness important when it is perfectly acceptable for an implementation to generate a result for a set of instructions that is close enough to one produced by a precise implementation? It is clear that the preciseness of today's computational model comes at a cost – a cost in the complexity of programming a solution, a cost in the verification of complex behavior specification, and a cost in the energy expended beyond the minimum needed to solve the problem.

We suggest here a model of computation that we term *approximate computing* to reflect the notion that there are computational and energy efficiencies to be gained by relaxing some of the strict rules at all levels. We believe that approximate computing could represent the right model for most computational needs of the future – activities such as decision support, search, and data-mining are consuming increasingly greater cycles of enterprise computing compared to traditional activities like accounting and inventory control. We provide below three examples to demonstrate the potential of approximate computing.

Today's computation platforms do not take enough advantage of knowledge from past computations either within the current run of a program or across instantiations of the program. Even when they do, for example in techniques such as "memoization", they expect the state of the machine to be precisely documented, and restrict prediction to situations where the state of the machine is identical to one that has been previously learnt. In typical table look-up schemes, detailed instruction execution is avoided only when the table contains an entry corresponding to a program counter value and when the entry is tagged with a machine state that is identical to the one at present; in these cases, the result of execution saved in the table is directly used. In approximate computing we would extend this approach so that even when a precise match does not exist, the system would be allowed to provide an interpolated solution, rather than stall and wait for the set of instructions to be precisely executed.

As a second example of approximate computing, we suggest allowing non-traditional behavior of existing hardware, especially under certain power-constrained situations. This may include substituting data types with ones having lower precision (e.g. double precision floating point with single precision), acceptance of slightly imprecise branch prediction (e.g. the number of loop iterations), or ignoring errors in results due to transient faults (e.g. in insignificant bits of data). Ideally, the programmer should be provided control over which computational results could be different and to what extent.

As a final example, we suggest that many real-world problems could be solved efficiently by building systems out of unreliable components and by fashioning algorithms that produce results that are accurate at higher granularity, though approximate at lower granularity. This is in contrast to the current approach, e.g. in the simulation of physical systems, where the results are exact at low granularity but approximate at the macroscopic level, due to constraints on hardware cost or execution latency. Nanotechnology, with its implicit unreliability of components, can be efficiently exploited only by tolerating the unreliability through architecture and software models that are aware of the unreliability, rather than by skirting around the unreliability and using conventional techniques.

There is an unprecedented amount of data being produced in the world today. Yet, cost and energy considerations are limiting a corresponding growth in the compute capability needed to process and analyze this data. It is likely that approximate computing will fulfill the processing needs of much of this new data being produced, especially by sensor networks. It is time for our community to begin designing new models for algorithms, architectures, and implementations to support approximate computing and help bridge this widening gap between the generation of data and the processing of this data in an energy-efficient manner.