Organic Computing

DISCLAIMER

Organic Computing

Anant Agarwal and Bill Harrod
MIT CSAIL and Darpa IPTO

August 2006

Imagine a revolutionary computing chip that can observe its own execution and optimize its behavior around a user's or application's needs. Imagine a programming capability by which users can specify their desired goals rather than how to perform a task, along with constraints in terms of an energy budget, a time constraint, or simply a preference for an approximate answer over an exact answer. Imagine further a computing chip that performs better according to a user's preferred goal the longer it runs an application. Such an architecture will enable, for example, a handheld radio or a cell phone that can run cooler the longer the connection time. Or, a chip that can perform reliably and continuously in a range of environments by tolerating hard and transient failures through self healing.

This paper proposes the vision of organic computation that will create such a self-aware computing device and an associated software system. An organic computer is given a goal and a budget – it then finds the best way to accomplish the goal with the means at hand. Much as in a biological organism, an organic computer has five major properties:

1. It is introspective or SELF-AWARE in that it can observe itself and optimize its behavior to meet its goals.
2. It is ADAPTIVE in that it observes the application behavior and adapts itself to optimize appropriate application metrics such as performance, power, or fault tolerance.
3. It is SELF HEALING in that it constantly monitors its resources for faults and takes corrective action as needed. Self healing can be viewed as an extremely important instance of self awareness and adaptivity.
4. It is GOAL ORIENTED in that it attempts to meet a user's or application's goals while optimizing constraints of interest.
5. It is APPROXIMATE in that it uses the least amount of precision to accomplish a given task. An organic computer can choose automatically between a range of representations to optimize execution – from analog, to single bits to 32-bit and 64-bit words, to floating point, to multi-level logic.

Organic computation can be distinguished from existing computational models which are largely procedural. Today's models require a user to specify a procedure of how something is to be done and the computer blindly follows this procedure irrespective of application or environmental conditions using a fixed set of prearranged resources. For example, if the user wants to use a computing device for software radio, then

the user programs it with a known bitwidth and prearranged code for algorithms such as Viterbi decode and FFT and to accept a given bitrate. The hardware is similarly fixed for all time. For example, the cache in the processing engine might be sized at 128Kbytes and two-way associative.

An organic computer, on the other hand, is given a goal and it attempts to achieve the goal with the minimal amount of resources and energy. Of course, it is also provided with many possible procedures to accomplish subtasks, each of which might use different types of architectural components. In our software radio example, the organic computer is given the goal of maintaining a connection to a receiver with a desired bit rate, using the least amount of energy. The software system and architecture collaborate on achieving this goal. An organic computer has cognitive hardware mechanisms in its trusted core to both OBSERVE and to AFFECT the execution. Since it is impossible to pre-configure all possible scenarios, the organic computer also implements learning and decision making engines in a judicious combination of hardware and software to determine the appropriate actions based on given observations. Thus, in our software radio example, the system will use the right precision for the FFT computations and the required amount of parallel hardware resources to achieve the goal. If the channel has very little noise, then the it might use a simpler coding scheme. The hardware will observe the execution of the code, and depending on the estimated working set size of the code, the system will shut off portions of the cache or make it direct mapped to save energy. At the same time, the system ensures that the goal is being met.

An organic computer can achieve 10x to 100x improvement in key metrics such as power efficiency and cost performance over extant computers. For instance, if for some streaming computation the system observes that 64 bits of precision is unnecessary (for example, if no changes are detected in the top 62 bits for a while) and can use 2 bits of precision, while at the same time turning off the data cache and using direct streaming of data over the network, the system can benefit from energy savings of 40x to 50x. As another example, the organic system might slow the clock to a sub module (and also the supply voltage) if its overall goal can be achieved with a much lower frequency. As a further example, in a tiled architecture running two streams of H.264 video encode, the system might observe the achieved output bandwidth for each stream, and move tiles between streams dynamically if the two video streams differ in complexity to maintain a fixed frame rate and a given per-stream bandwidth requirement.

Probably much more importantly, much like biological organisms, an organic computer can go well beyond traditional measures of goodness like performance and can adapt to different environments and even improve itself over time. It can also perform ”code intrusion detection” by flagging abnormal behavior in its software by learning and maintaining signatures of its normal behavior. Corrective action might include shutting itself down or in some cases applying self healing. In doing so, the organic computer can build upon technologies developed for systems in the previous intrusion tolerant systems (ITS) program out of IPTO in which a congruence between self healing for faults and for malicious intrusions was demonstrated.

Why now? Although such a machine may seem rather far fetched, we believe that basic semiconductor technology, computer architecture and software systems have advanced to the point that the time is ripe to realize such a system. To illustrate, let us examine each of the key aspects of organic computation including introspection, approximation, goal orientation, adaptation and self healing. We will discuss how they might be built in a practical way, and identify the fundamental challenges that we will have to overcome.

Introspection or self awareness implies that the system can observe itself while it is executing. The processor hardware can include mechanisms to observe instantaneous cache miss rates, bit positions in data words that are changing, cache sets that are hot versus others that are idle, numbers of errors in data transmissions or memory access, branch directions, network and memory latencies and queue lengths, among many

others. These measurements will feed adaptation mechanisms that will adapt the architecture as needed. Introspection or self awareness requires foundational changes to computer architecture - self aware computers need mechanisms to observe themselves. Fortunately, semiconductor technology makes available billions of transistors on a single chip, so throwing transistors at the problem of building observers and recording state is eminently feasible today. Our challenge will be to identify what metrics are worth observing, how to make the measurements without impacting the execution, and what we can do with the results. We have some existing examples in this area. In recent work, we have shown that we can observe phase changes in program execution and change the cache access hash function to optimize cache miss rates. We have demonstrated that cache miss rates can be halved for many applications in this manner. In another body of work related to tiled architectures, we have used an adjacent helper tile to observe the execution (in particular, memory reference patterns) of a given master tile and prefetch data into the master tile's cache before it is needed.

Approximate computation implies that the computer does not always use the most available precision to accomplish a task. For instance, modern day processors have reached 64 bits of digital data widths. This data width is used in all computations whether it is needed or not. There are many classes of computations for which this precision is overkill. As an extreme example, imagine an image recognition task in which the final answer to the user is a single bit - yes or no (for example, is there a tank in this image or not). It is quite possible that a simple edge representation using just one bit per pixel might suffice to perform the pattern recognition task. Approximation can be applied in many other areas of digital design as well, and in fact, we question the very basic overkill digital design philosophy, viz, requiring signals to be fully restored after each logic element. The computer can try to use the minimal precision and probably even multilevel logic or the analog representation in its computations to produce a result. One research challenge with approximation is to figure out the minimal precision needed for a given computation. Another challenge will be to discover the best way to introduce analog representation into what is fundamentally a digital computer. Some recent work in this area includes compiler supported bit-width analysis. Other work that directly applies here is that of Rinard et al which has shown the possibility of highly reliable computation even when erroneous data values are ignored and allowed to propagate during program execution.

Goal orientation implies a revolutionary transition in architecture and algorithm design from a procedural style of specification to a goal oriented style. Goals indicate precisely what the user wants, not how to get there. This way, the computer can determine how best to achieve a user's goals depending on the conditions on the ground. Goal orientation can be applied at all levels of a system – from the specification of the application all the way down to transmission of bits on a wire. In the latter case, a communications channel within the chip might choose to perform lossy compression to achieve effectively higher bandwidth transfer if the goal of the higher level application does not care about an exact representation. An example of an architectural goal can be to maintain no more than a maximum bandwidth demand on the memory system. An example of a system goal might be to maintain a given maximum power dissipation. Recent work along these lines includes the GOALS system that is a software system that attempts to meet user-driven goals, rather than follow set procedures.

Adaptation is the ability of the computer to change what it is doing or how it is doing a given thing at run time. A key part of adaptation is the development of a control system as part of the computer architecture that observes execution, measures thresholds and compares them to goals, and then adapts the architecture or algorithms as needed. A key challenge is to identify what parts of a computer need to be adapted and to quantify the degree to which adaptation can afford savings in metrics of interest to us. Examples of mechanisms that can be adapted include various cache parameters such as associativity and replacement

algorithm, prefetch methods, number of tiles used in a computation, and the presence or absence of coding or compression when transmitting data. Recent research on reactive synchronization is another example of adaptation in which the waiting algorithm was tailored at run time to the observed delay in lock acquisition.

Self healing is an extremely important special case of adaptation. We give it independent billing because in the future era of multiple billions of transistors on a chip and deep submicron technologies, continuous correct operation in the presence of transient and hard failures will become a basic requirement. Thus a self healing system can use introspection to observe where errors are occurring and perform appropriate adaptation to fix the problem. For example, if errors are seen during data transmission on a given link, then the system can use one of two mechanisms to self heal. (1) It can use introduce coding to correct errors, or (2) it can cause messages to be rerouted to bypass the faulty region. The same technique can be used in caches to turn off cache banks that are producing errors.

Much like in the Darpa PCA program, organic computation applies to all levels of a computer system, and will involve research in computer architecture, VLSI chip design, runtime software systems and compilers.

We will also establish measures of success for this program. Some examples of precise measures of success include:

- Build a chip which demonstrates less than half the initial power dissipation after a period of observation for a given application.
- Build a chip which demonstrates twice its initial throughput after an initial observation period for a given application.
- Build a chip that maintains a given throughput goal, for example, for a transactional workload, even when the external DRAM speeds are changed by a factor of 10.
- Build a system containing a pair of organic computing chips working together on a board with a wired channel connecting them. The system continues correct operation, perhaps with a graceful performance degradation, even as some of the wires connecting the two chips are purposefully cut.

The computing industry is at a major crossroads. Semiconductor technology offers tens of billions of transistors on a chip, and future advancements show no signs of abating. In recent times, unfortunately, these advancements have not resulted in proportional increases in performance or other measures of interest to users. Thus, the computing industry is ready and receptive to the next major revolution in computer architecture. We believe that the proposed vision on organic computation can revolutionize computing from a procedural model to a goal oriented world in which programmers deal with select goals, not with each possible case, thereby reaching unprecedented levels of productivity, performance, and resilience.