Self-Organizing Multimedia and Extremely Distributed Architectures (Keynote Presentation)

William Butera and V. Michael Bove, Jr.

MIT Media Laboratory, Cambridge MA USA
{bill, vmb}@media.mit.edu, http://www.media.mit.edu/~vmb/obmg.html

Abstract. The Object-Based Media Group at the MIT Media Laboratory is developing robust, self-organizing programming models for dense ensembles of ultra-miniaturized computing nodes which are deployed by the thousands in bulk fashion, *e.g.* embedded into building materials. While such systems potentially offer almost unlimited computation for multimedia purposes, the individual devices contain tiny amounts of memory, lack explicit addresses, have wireless communication ranges only in the range of millimeters to centimeters, and are expected to fail at high rates. An unorthodox approach to handling of multimedia data is required in order to achieve useful, reliable work in such an environment. We describe the hardware and software strategies, and demonstrate several examples showing the processing of images and sound in such a system.

1 Introduction

We begin by observing that semiconductor process technology will shortly arrive at the point where autonomous computing elements – possibly coupled with sensing or actuating devices – can be scaled to the size of large sand grains and sold at bulk prices. It then will become possible to move computing out of the sort of packaging in which it is currently housed and make it part of the built environment, embedding processors in rboards, and into everyday objects such as furniture, clothing and random surfaces. As a representative embodiment, consider the architecture advanced by Sussman, Abelson and Knight [1]: ultra-miniaturized computing nodes each fitted with an on board microprocessor, 50K of memory and a wireless transceiver, all shrunk down to the size of a pin head and powered parasitically.

In the domain of interest, these nodes would be embedded in a 2D surface (such as the plywood in a table top) with a density on the order of tens of thousands per square meter. Positioning would be pseudo-random, with no local restrictions on density or regularity. Once exposed to power, they should boot and self organize their local address space. External I/O would be via physical proximity with an object fitted with a transceiver whose protocols are identical to the transceivers on the chips.

In our research we adopt a hardware reference model constructed around a single IC with dimensions 2 mm x 2 mm. Onboard subsystems include a block for power harvesting, a full featured microprocessor, a wireless transceiver for inter-particle communication within a radius of at most a few centimeters, a 50 MHz internal clock, a ROM for the operating system, and approximately 50K RAM for program and data space. More detail on the hardware characteristics of the proposed system can be found in [2] and [3].

2 Software Model

Such an ensemble of processing nodes, running asynchronously (and perhaps intermittently) and communicating locally via an ad hoc network places unusual demands on the programming model. Worst among these are:

- *Asynchrony:* Clock level synchrony is out of the question. Two neighboring particles cannot be guaranteed to have the same clock rate, let alone phase lock. Event level synchrony also seems beyond reach. In an unknown topology with sporadic unit failures, there is no way for a process on one particle to predict what processes will be running on a neighboring particle. Code running on one particle should never explicitly synchronize to events generated on another particle.
- *Extreme Fault Tolerance:* Allied with the inherent asynchrony is the propensity of individual particles to fail completely, whether because of lack of power, hardware problems, interference with communications, or physical damage (*e.g.* hammering a nail into a surface filled with these processors may disable one or more.)
- *Network Locality:* Particles can communicate directly only with other particles in the immediate spatial vicinity. While the size of the neighborhood can vary substantially, current experiments run on neighborhood sizes ranging from 8 to 20 particles.
- *Adaptive Topology:* Any ensemble of embedded particles will have final topology which is unknown at the time that much of the application code is written. While it will always be possible to recover an approximate coordinate system at run-time, no application code should rely on a particular spatial layout of the processors.
- *Code Compactness:* On-particle memory is very limited, inter-particle bandwidth is slow compared to processor speed, and there is no external support for virtual memory. Functions running on a given particle should therefore be self-contained and sized to fit completely in a single particle.
- *Shared Data:* Nevertheless, the utility of a single particle's computation will often go up if it has access to results from local computations on neighboring particles. With the caveat that no process can predict what processes are running in the neighborhood, tagged data passed within the neighborhood should be available to processes running on a given particle.
- *Mobility:* Inter-particle migration of code segments will increase both the functionality of the individual particles and the adaptability of the overall

system. The restriction here is that exact trajectory of the migrating code cannot be pre-ordained.

Programs running on a particle's microprocessor reside in the particle's RAM space. Most of the RAM is available for use as program, data, and scratch space for these programs. However, a section of the RAM is reserved for what we refer to as the I/O space – an area which is at least readable by any program running on the particle's microprocessor. A subset of the I/O space is called the HomePage. The HomePage is an area where programs can both read and write tagged data. Any program local to the particle can post to the HomePage, and posts to the HomePage are readable by all local programs.

The remainder of the I/O space is subdivided into mirrored instances of the HomePages of neighboring particles. When a program on a given particle posts a piece of tagged data to the particle's HomePage, copies of that post appear at the mirror sites of all the neighboring particles. The caveat is that the latency in the mirroring operation is unconstrained.

Collectively, the I/O space functions as a public bulletin board, where the HomePage portion is writable and the entire I/O space is readable.

At run-time, code segments migrate nomadically looking for particles on which to install themselves. In those particles where entry into the program RAM is successful, the code segments will set up shop and begin searching for relevant data in the I/O space. The side effect of the code segment's activities is additional posts to the HomePage. Often, the number of code segments seeking entry will exceed the particle's capacity. The allocation of program space is regulated by the operating system (OS) in response to competition among the code segments. Each code segment must draw its competitive advantage from the I/O space and therefore, indirectly from the activity of other code segments. The competition is arbitrated by the particle's OS. And when a particular code segment loses out, it is de-installed and passed to the output port to migrate further via diffusion.

As a platform for application development, we have developed a simulator modeled after the Gunk simulator [4]. Code segments are written as Java objects with the functions for communicating with the OS implemented as public methods. Each segment is archived individually as a serialized Java object. I/O portals, capable of mimicking the networking behavior of the particles, can be arbitrarily positioned to diffuse the code segments into the particle ensemble. Ilustrations to follow are snapshots taken from the simulator environment.

How does a particle in one location get data to particles more distant than its direct communications radius, given that there is no explicit addressing? A method that we have found useful is to diffuse a code segment that generates a gradient field tagged with a label identifying the source (Figs. 1,2). It is then possible for code segments in remote locations to "ride the gradient uphill" to communicate with the source, or indeed to migrate back in that direction. More details can be found in reference [2].

3 Multimedia Applications

The use of such a system for storage or processing of streaming multimedia data is complicated by the lack of explicit addressing. Data packets must therefore themselves know where to go and to make this decision dynamically in response to the topology of the particle ensemble.

An example application we have implemented on the simulator stores packetized audio data in the memory of a particle ensemble. Data is exchanged with the particle ensemble via streaming through arbitrarily positioned I/O portals. On input, the audio packets should diffuse outward, quickly distancing themselves from the input portal. Once the input streaming is complete, the packets should uniformly distribute themselves throughout the ensemble's collective memory. In the process, they should also randomize their position, effectively decorrelating the time codes of the packets from their spatial position. Finally, on output, the packets should reestablish their original sequential ordering prior to streaming out through an output portal.

During pre-processing, audio is packetized and each packet is embedded into a code segment called a Carrier. The Carriers also record a stream-relative time code for the packet. Once the Carriers are streamed into the particle ensemble, the transport behavior of the audio packets is defined by the migration strategy of the Carrier. At the start of every execution cycle, the Carriers determine whether they are diffusing through the ensemble or being retrieved for playback. In the diffusion mode, the Carriers migrate randomly, yielding a quasi-uniform distribution of the Carriers over the entire particle ensemble, and a spatial shuffling of the Carriers. This is the default mode and is active during storage. Carriers enter their call-back mode when they see a gradient radiating from an output portal. The posts from this gradient field list estimates of the distance back to the output portal, an ID for the requested audio stream, and range of time codes which are active and should soon be queued for playback. When a Carrier sees the post from the CallBackGradient, it does one of three things: if its time code falls within the range of active time codes, it proceeds directly toward the output portal; else if it is too close to the portal, it moves away from the gradient source, thus increasing the bandwidth efficiency in the vicinity of the portal; otherwise, it builds a local average of time codes, and adjusts its position relative to this average using the gradient field for orientation.

Using a similar diffusive strategy, one can use this system to store images. If the images are subjected to a hierarchical wavelet decomposition and the more important coefficients are replicated in multiple packets, even relatively small subsets of the particle ensemble can yield usable (if imperfect) reconstructions of the images.

4 Acknowledgments

The authors wish to thank numerous co-workers from the faculty, staff, and student body at the MIT Media Laboratory and the MIT Laboratory for Computer

Science. Particular thanks to Joshua Lifton, Stefan Agamanolis, Chris Hanson, and James McBride. This work has been supported by the Digital Life Consortium and by a fellowship from Intel corporation.

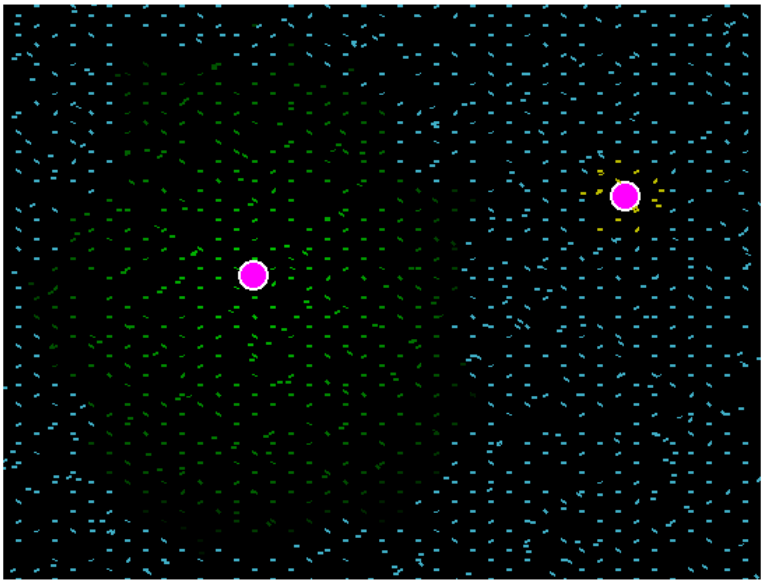

Fig. 1. Snapshot of the simulator, showing code diffusing outwardly from the source on the left.

References

1. H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss: Embedding the Internet: amorphous computing. Communications of the ACM **43(5)**, (2000).
2. W. Butera: Programming a Paintable Computer. PhD thesis, Massachusetts Institute of Technology, Cambridge MA USA, (2001).
3. W. Butera and V. M. Bove, Jr.: Literally Embedded Processors. Proc. SPIE Media Processors **4313**, Society of Photo-Optical Instrumentation Engineers, Bellingham WA USA, (2001).
4. S. Adams: A high level simulator for Gunk. Technical report, Massachusetts Institute of Technology AI Lab, Cambridge MA USA, (1997).

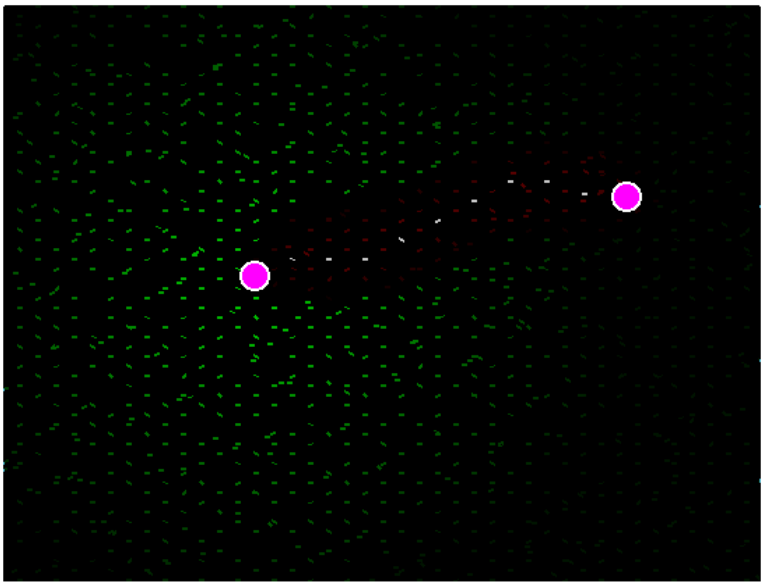

Fig. 2. The portal on the right can now route information to the portal on the left by following the gradient.