# Extremely Distributed Media Processing

William Butera, V. Michael Bove, Jr., and James McBride
MIT Media Laboratory, Cambridge MA USA

## 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.

Keywords: Media processors, self-organizing systems, networks

## 1. BACKGROUND / HARDWARE MODEL

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 in bulk rather than by the unit. 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 building materials, and into everyday objects such as furniture, clothing and random surfaces. As a representative embodiment, consider the architecture proposed by Sussman, Abelson and Knight[1]: ultra-miniaturized computing nodes each fitted with an on board microprocessor, 50 Kbytes of memory and a wireless transceiver, all shrunk down to the size of a pin head and powered parasitically.

In our scenario, 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 (486-class), a wireless transceiver for inter-particle communication at a minimum of 100 Kbytes/second within a radius of at most a few centimeters, a 50 MHz internal clock, a ROM for the operating system (OS), and approximately 50-100 Kbytes RAM for program and data space. We assume that each particle can communicate at most with only ten to twenty other particles. Once exposed to power, each particle builds an enumerated list of its neighbors with whom it can communicate. There is no hardware support for recovering relative orientation or distance. And critically, no particle has any knowledge of the world beyond its communication radius. More detail on the hardware characteristics of the proposed system can be found in references [2] and [3].

## 2. SOFTWARE MODEL

Software on a such a system is organized into autonomous, self-contained executables referred to as “process fragments.” Fragments 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.

The size of the local HomePage is determined by the particle's OS, which de-fragments the HomePage as necessary. The OS also dynamically maintains the size of the I/O space. For every neighbor with which a particle maintains an active contact, the OS allots a mirror site in the local I/O space for the neighbor's HomePage. The constituency of the neighborhood is periodically checked, and changes in the number of active neighbors triggers an adjustment in the number of mirror sites. Thus the overall system will continue to function if individual particles become active or inactive, or if new particles come into proximity (thus an external device can communicate with the system by emulating – or literally containing – a particle).

At run-time, process fragments migrate nomadically looking for particles on which to install themselves. In those particles where entry into the program RAM is successful, the process fragments will set up shop and begin searching for relevant data in the I/O space. The side effect of the process fragment's activities is additional posts to the HomePage. Often, the number of process fragments seeking entry will exceed the particle's capacity. The allocation of program space is regulated by the operating system in response to competition among the process fragments. Each process fragment must draw its competitive advantage from the I/O space and therefore, indirectly from the activity of other process fragments. The competition is arbitrated by the particle's OS. And when a particular process fragment loses out, it is de-installed and passed to the output port to migrate further via diffusion. Note that in this system there is no such thing as a packet of pure data; everything to be transported or processed must be packaged within a process fragment.

## 3. MULTIMEDIA APPLICATIONS

While hardware work proceeds, we have developed a simulator modeled after the Gunk simulator[3] as a platform for software development. Process fragments are written as Java objects with the functions for communicating with the OS implemented as public methods. Each fragment 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 process fragment into the particle ensemble. Illustrations 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 process fragment that generates a gradient field tagged with a label identifying the source (Figs. 1,2). Essentially, it does this by sending copies of itself to "uninfected" particles, incrementing an integer number of hops back to the source; then each copy estimates a floating-point distance based upon the integer hop counts of all its neighbors. It is then possible for process fragments in remote locations to "ride the gradient uphill" to communicate with the source, or indeed to migrate back in that direction. "Halo" process fragments dynamically construct a mini-gradient field about the markers along the path; in the case of multiple point to point connections, the halo acts to inhibit other marker trails from crossing each other. More details can be found in reference [2].

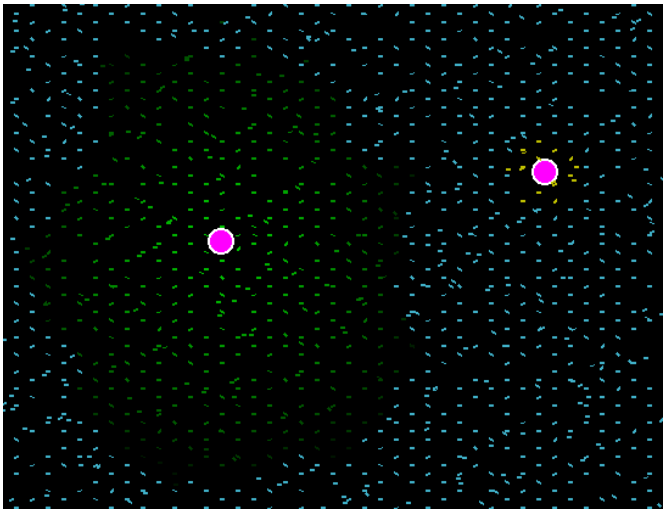

**Figure 1: Snapshot of the simulator, showing code diffusing outwardly from the source on the left. Particles are color-coded according to their estimate of distance from the source.**

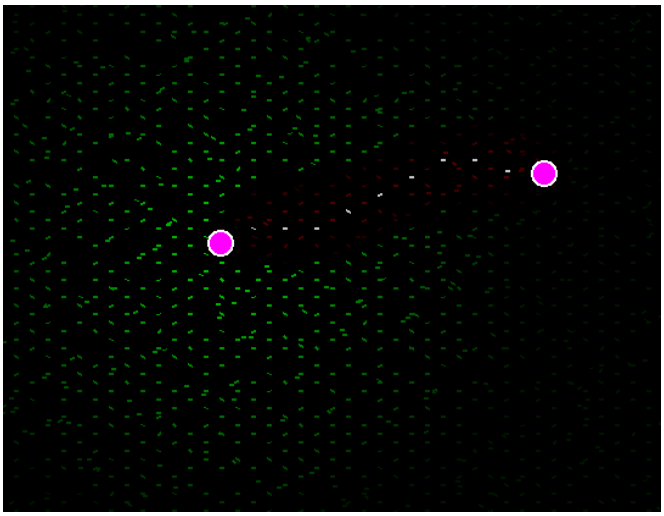

**Figure 2: Creation of an "insulated" communication pathway between a source and a destination.**

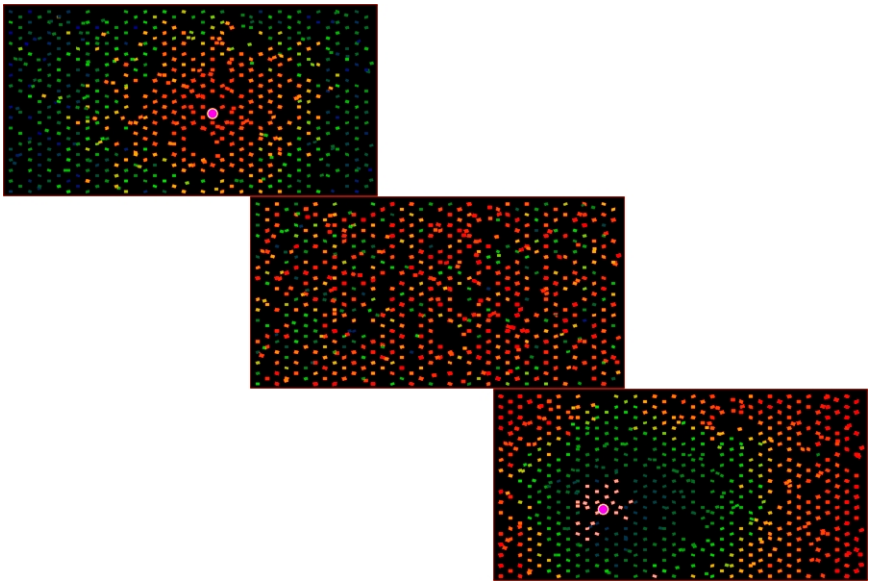

**Figure 3: Storage and playback of packetized audio. (Top) Audio packets stream into the surface from the portal to the right of center. (Center) The packets randomize their locations. (Bottom) Audio packets stream out of the surface into the portal to the left of center.**

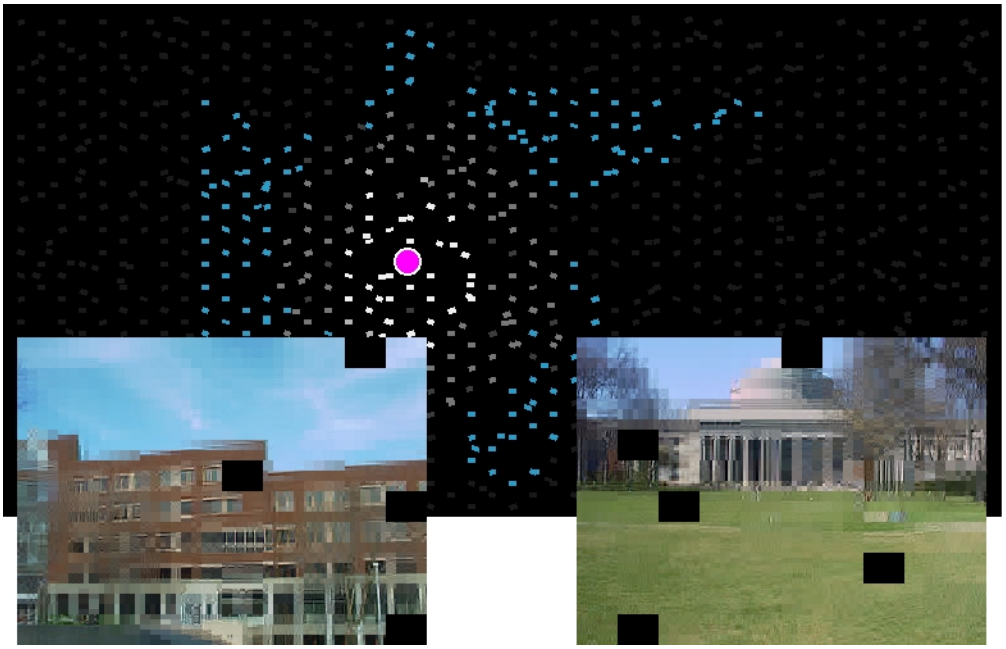

**Figure 4: After several images have been wavelet-decomposed and sets of their coefficients diffused throughout the surface, we can reconstruct (possibly imperfect) pictures from just a subset of the particles.**

In the gradient example, while each process fragment is executing a simple strategy over a restricted locality, the global effect is a dynamically configurable communication network. More promising than this simple example are applications which take advantage of the fact that each of the links in the communication chain is itself a computational device, and can perform useful tasks on the data passing through, pulling in additional computation and information from the surroundings as needed. Several are in development as of this writing, with a particular emphasis on media 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 must 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 positions – otherwise, the "pressure" of the incoming packets would tend to push the earliest-arriving (and earliest-needed on playback) packets far from the source portal. Finally, on playback, the packets should reestablish their original sequential ordering prior to streaming out through an output portal (Fig. 3).

During pre-processing, audio is packetized and each packet is embedded into a process fragment 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 (Fig. 4).

Image storage is implemented as the interaction between two process fragment types: Carriers and Transforms. Prior to entry into the particle ensemble, an image is divided up into blocks, and each block is embedded in a separate Carrier. The Carriers are streamed into the particle ensemble where they spread via diffusion. A single Transform process fragment is streamed into the ensemble and propagates a single copy of itself to every particle. Transforms read the image data from Carriers and apply a block frequency transformation. The transformation uses a 2-tap Haar kernel recursively applied to produce a 3-level pyramid. The output of this transformation is 10 subbands, each of which isolates energy at a selected orientation and frequency. The subbands are read by the Carriers which then replace their image payload with the transform data. Once a Carrier has replaced its space domain payload with a frequency domain payload, it splits itself up into 9 small mini-Carriers. The payload for each mini-Carrier consists of two subbands: a copy of the lowest frequency subband together with one of the 9 remaining higher frequency subbands. The nine mini-Carriers then diffuse evenly among the particles.

Image segmentation is a problem whose topology is well matched to that of a 2D particle ensemble, provided that there is correspondence between the actual image and the spatial locations of the particles (for example, if some of the particles themselves incorporate sensors and an optical system projects the image onto the surface). The two crucial elements are the ability to deliver the input data in parallel, and the ability to structure the processing for concurrent operation on a irregular lattice. We have begun experiments in which we apply a "competition among experts" paradigm[4] with multiple process fragments in a neighborhood estimating the MAP (maximum a posteriori) likelihood that a pixel belongs to a particular object. These estimates are reported in HomePage posts and the decision is made using a simple *max()* function. Experts can be diffused into the system in some sequence and the segmentation can improve as each arrives.

## 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 and Chris Hanson. This work has been supported by the Digital Life Consortium and by a fellowship from Intel corporation.

## 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, 2001.
3. W. Butera and V. M. Bove, Jr., "Literally embedded processors," *Proc. SPIE Media Processors 2001*, **4313**, pp. 29-37.
4. V. M. Bove, Jr. and W. Butera, "The Coding Ecology: Image Coding Via Competition among Experts," *IEEE Transactions on Circuits and Systems for Video Technology*, **10**, October 2000, pp. 1049-1058.