Programming a Paintable Computer

William Joseph Butera
SBEE, MIT, 1982
SM, MIT, 1988

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning, in partial fulfillment of the
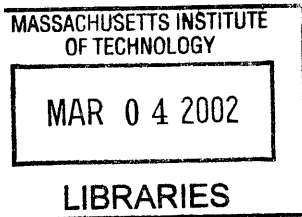requirements for the degree of

Doctor of Philosophy in Media Arts and Sciences

at the

Massachusetts Institute of Technology

February 2002

Author _____
William Joseph Butera
Program in Media Arts and Sciences

Certified by _____
V. Michael Bove Jr.
Principal Research Scientist
Program in Media Arts and Sciences

Accepted by _____
Andrew Lippman
Chair, Departmental Committee on Graduate Students
Program in Media Arts and Sciences

Programming a Paintable Computer

Bill Butera

Submitted to the Program in Media Arts and Sciences, School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy.

at the

Massachusetts Institute of Technology

February 2002

Abstract

A paintable computer is defined as an agglomerate of numerous, finely dispersed, ultra-miniaturized computing particles; each positioned randomly, running asynchronously and communicating locally. Individual particles are tightly resource bound, and processing is necessarily distributed. Yet computing elements are vanishingly cheap and are regarded as freely expendable. In this regime, a limiting problem is the distribution of processing over a particle ensemble whose topology can vary unexpectedly.

The principles of material self-assembly are employed to guide the positioning of "process fragments" — autonomous, mobile pieces of a larger process. These fragments spatially position themselves and re-aggregate into a running process. We present the results of simulations to show that "process self-assembly" is viable, robust and supports a variety of useful applications on a paintable computer.

We describe a hardware reference platform as an initial guide to the application domain. We describe a programming model which normatively defines the term process fragment and which provides environmental support for the fragment's mobility, scheduling and data exchange. The programming model is embodied in a simulator that supports development, test and visualization on a 2D particle ensemble.

Experiments on simple combinations of fragments demonstrate robustness and explore the limits of scale invariance. Process fragments are shown interacting to approximate conservative fields, and using these fields to implement scaffolded and thermodynamic self-assembly. Four applications demonstrate practical relevance, delineate the application domain and collectively illustrate the paintable's capacity for storage, communication and signal processing. These four applications are Audio Streaming, Holistic Data Storage, Surface Bus and Image Segmentation.

Thesis Supervisor: V.Michael Bove Jr.
Principal Research Scientist, Media Arts and Sciences, MIT

Dissertation Committee

Thesis Supervisor:

V. Michael Bove Jr. _____
Principal Research Scientist
Media Arts and Sciences, MIT

Thesis Readers:

Edward Adelson
Professor
Department of Brain and Cognitive Sciences, MIT

/

Neil Gershenfeld
Associate Professor
Media Arts and Sciences. MIT

Gerald J. Sussman _
Professor
Department of Electrical Engineering and Computer Science, MIT

Acknowledgments

Even ordinary people like me can have extraordinary ideas.
But it helps to have them while you are at MIT.

My heartfelt thanks to ...

To my advisor, V. Michael Bove Jr., long a steadfast critic, ultimately a guiding hand, ever an approachable collaborator. Mike's interest in machine architecture runs deep. And the many hours we spent debating the relative merits of distributed versus centralized architectures crystallized the need to qualify this work early with simulated applications.

To my thesis readers, Neil Gershenfeld, Gerald Jay Sussman and Edward Adelson; masterful and selfless educators all, each with something fundamental to say about the nature of information and computation. Conversations with these men were empowering, refreshing, provocative, and always instructive. It was a series of meetings with Neil 27 months ago that launched this project, complete with conceptual grounding, articulated purpose and crucial support. Ensuing contact with Gerry and the amorphous computing group added background, a sense of shared purpose and the kind of litmus tests that can only come from a progenitor of the field. As both an accomplished vision scientist and a skilled generalist, Ted added additional focus on questions relating the basic nature of computation, (as opposed to ways to simply build a better computer). For this, and his vital sanity checks on the image segmentation application, I am especially grateful.

If this idea blossomed at MIT, it was conceived at Micronas-Intermetall, a mid-sized German IC manufacturer with a over-sized pioneering spirit. It was there, over years of lunch time brain-dumps, that the idea of a paintable computer began to take shape, anchored on the aspirations of a world class troupe of intrepid chip hackers. Hats off to Lubo, Sönke, Uli 7, Herbert, Bini, Christian, Milan, Martin, Werner, Hans-Jürgen, Heinrich, Rehmi, Uli S, Manfred, Dietmar, Kai, Klaus, Dieter, Stefan, Thomas, Peter, Fritz, Knut, and Dagmar. With them in mind, this document takes the form of a 170 page letter from camp — back to the industry that launched me on this adventure, and in particular to the Concept Engineering department at Micronas ... happy home for over six years.

Not surprisingly, this research owes an abiding debt of gratitude to my department; the Media Laboratory at MIT. To Nicholas Negroponte, founding director of the lab, for his skill, acumen, generosity and unshakable faith in the students' inner compass. To Walter Bender, current guardian of the MAS flame, for his early enthusiasm, penetrating insights and bulwark support during the formative phase of this research. To Linda Peterson for her fortitude, patience, administrative wizardry, and dedication to the well being of her charges. To Tom Gardos from Intel, and Mike Taylor from Motorola, for their masterful stage management of what ultimately became profoundly rewarding interactions with their respective companies. And finally, to my fellow students, particularly those in the Garden, and the Physics-and-Media groups, for their advice, inspiration, fellowship, patience, expertise, humor, and generational diplomacy. The world will be in good hands.

Contents

Contents

Introduction

Scenario - Painting the Computing

In the next years, process technology will arrive at the point where autonomous computing elements can be scaled to the size of large sand kernels and sold at bulk prices. Coupled with a commensurate shrink in the footprint of sensors and actuators, the concept of "personal computing" will take on a radically new dimension. While the details of how people relate to this ultra-commoditized form of computing remain largely conjectural, a couple of points are already apparent:

1. As the computing elements become resilient to environmental stress, they will migrate off the expensive, precision engineered motherboards, and into everyday objects such as building materials, furniture, and clothing.

2. People will find it more natural to deal with computation as a bulk item, preferring to manipulate it by the jar full, by the bolt, by the cord, or by the shot glass.

One could loosely delineate commodity level computing as those instances where the price of the computing is so low that it is comparable to detergent and where the form factor is so small that it seamlessly blends into everyday environment.

As a representative embodiment, we advance the notion of a *paintable computer*. Based on the architecture originally described by Sussman, Abelson and Knight [1], a pinless IC with an on board micro, program memory and a wireless transceiver, is reduced to the size of a small match head and powered parasitically. Several thousand of these particles would be suspended into a viscous medium and deposited it on surfaces like paint. Once exposed to power, they should boot and self organize their local address space. External I/O would be via physical contact with an object fitted with a transceiver whose protocols mimic the behavior of the chips.

While the details will change en route to practice, this notion of a *paintable* captures the essence of what could be a big part of our computing future: computation as a tangible, fluidly dispersible additive to ordinary objects. Want a surface to be smart? Add a layer of computing. Want it to be smarter? Add a second coat. Has the computing lost its luster? Get out the belt sander.

The atomic element of a *paintable* is the particle (fig 1-1*a*). Characteristic specs include a '486 class micro, an internal clock running at ~ 50 MHz, and 50-100K of RAM for code/data storage. All the I/O to the micro is gated through a wireless transceiver supporting a minimum duplex rate of 100 Kb/s.Communication is via asynchronous links to the nearest neighbors. A power subsystem harvests power from the immediate environment with minimal constraints on the particle's placement[1].

Once exposed to power, each particle builds an enumerated list of the neighbors with which it can communicate (fig. 1-1*b*). 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.

---

1. Candidate techniques include chemical, optical, electrodynamic coupling or sliding mechanical contact to conductive planes.

(a)

(b)
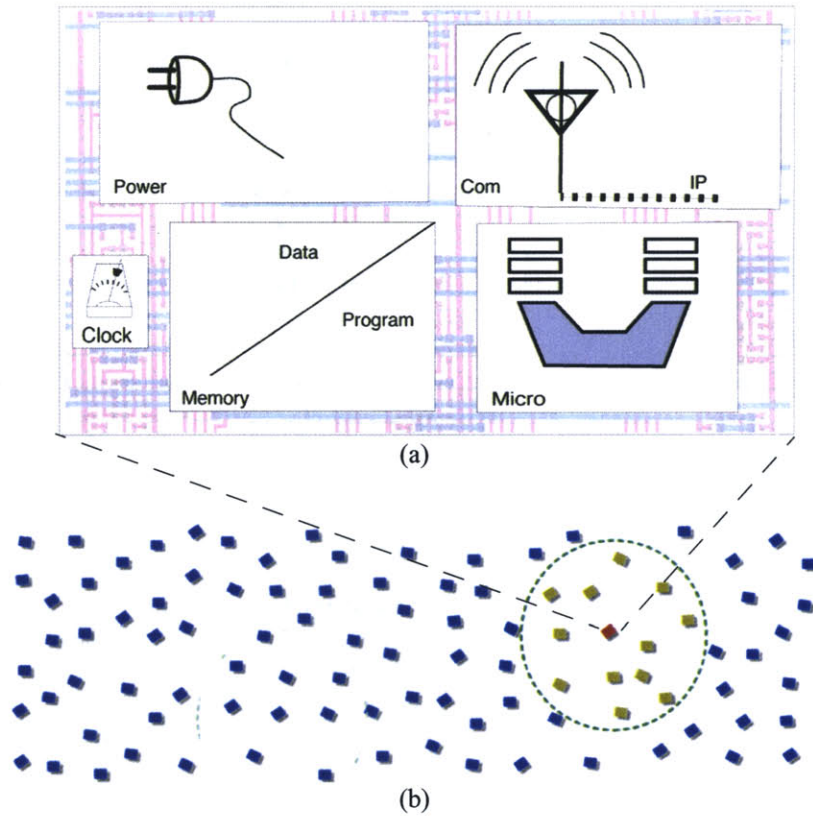
FIGURE 1-1. Paintable computer: particle and ensemble

Paintable computers are ensembles of homogeneous particles (a) each containing a micro, memory, a wireless transceiver and support for harvesting environmental power.

Particles are pseudo randomly positioned and communicate locally with their immediate neighbors (b). On boot-up, particles build an enumerated list of their neighbors . There is no hardware support for estimating distance or orientation. And particles are purposefully blind to the world beyond their communication radius.

The Problem - Programming Paintchips

Decades of research and productizing have failed to produce robust, general methods for programming massively parallel systems that could take hold in the marketplace. And the native architecture of a *paintable* seems to take a bad thing to a fatal extreme. Indeed the basic attributes read like a compiler designer's epitaph; *an unknown number of micros arranged in an unknown topology with slow, asynchronous local interconnects. Individually, each micro is too resource poor to do any useful work, yet the network message flow is chaotic and the unit reliability is low.* Here is a detailed look at the worst of these sorrows:

Asynchrony Clock level synchrony is out. Two neighboring particles can not be guaranteed to have the same clock rate, let alone lock them. 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 or direct the activity on a neighboring particle. Code running on one particle should therefore 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. A defining characteristic of a paintable computer is that the user should be permitted certain activity that will cause some particles to die. For example, if a *paintable* is layered onto a wooden surface, the user should think nothing of driving a nail into that surface, or machining it to an arbitrary shape.

Network Locality Particles can only communicate directly with other particles in the immediate spatial vicinity. A particle's knowledge of the environment stops completely at the border of this neighborhood. And even within this neighborhood, particles have no sense of relative orientation or distance to neighbors[2].

Adaptive Topology Any truly *paintable* system will have final topology which is unknown at the time when the application code is written. While it will always be possible to recover an approximate coordinate system at runtime, no application code should prescribe a particular spatial layout of the processors. As a consequence, application code may not explicitly address a

---

2. While the size of the neighborhood can vary substantially, current experiments run on neighborhood sizes ranging from 8 to 20 particles.

particle by location   neither as an absolute location nor as a relative location (eg. two hops north).

Code Compactness On-particle memory is very limited, inter-particle communication 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.

While there is nothing to prevent particles from passing data or code to their neighbors, no process on a given particle can predict or control the state of processes running in the neighborhood.

Combinatorics A basic strategy of compiler design for massively parallel machines is to anticipate likely failure modes and adapt the data flow to optimize for speed and tolerance to foreseeable faults. This in turn imputes to the compiler designer the ability to predict and account for all possible hardware-related states surrounding a particular computational event.

For systems where the native combinatorics outstrip the compiler's ability to predict and adapt, the only option is to impose those restrictions necessary to limit the number of possible states. In many ways, the *paintable*, with its unconstrained placement of particles, represents a worst case in the combinatorics of the hardware and would require the largest number of restrictions to tame its complexity.

The Solution - Self-assembly

Cumulatively, these hurdles suggest that it will never be practical to expect a human to structure a procedure for unrestricted use on this architecture. The starting point for this research is the claim that *if we can not get a human to structure the procedures, we are going to have to get the procedures to structure themselves.* Consequently, this work advances a programming model based on process self-assembly — the undirected reassembly of a process from randomly distributed fragments of code with state.

A ready analog to this notion can be found in the self assembly of materials. Self-assembly in the material domain is defined as "*the spontaneous organization of objects, under equilibrium conditions, into stable aggregates*"[7]. Investigators

have reported measured success at assembling complex irregular structures from suitably treated component parts [22][29]. Often this assembly process is reversible, suggesting that one could start with a end structure, deconstruct it into its original parts, place the parts into a medium which supports mobility, and agitate them until the reassembly process achieves a local minimum — preferably in the original state (fig. 1-2)

FIGURE 1-2. Material re-assembly

The original aggregate structure (a) consisting of six components is broken up into these individual parts (b). Once placed into a fluid medium and agitated, the parts incrementally reassemble (c) (d).
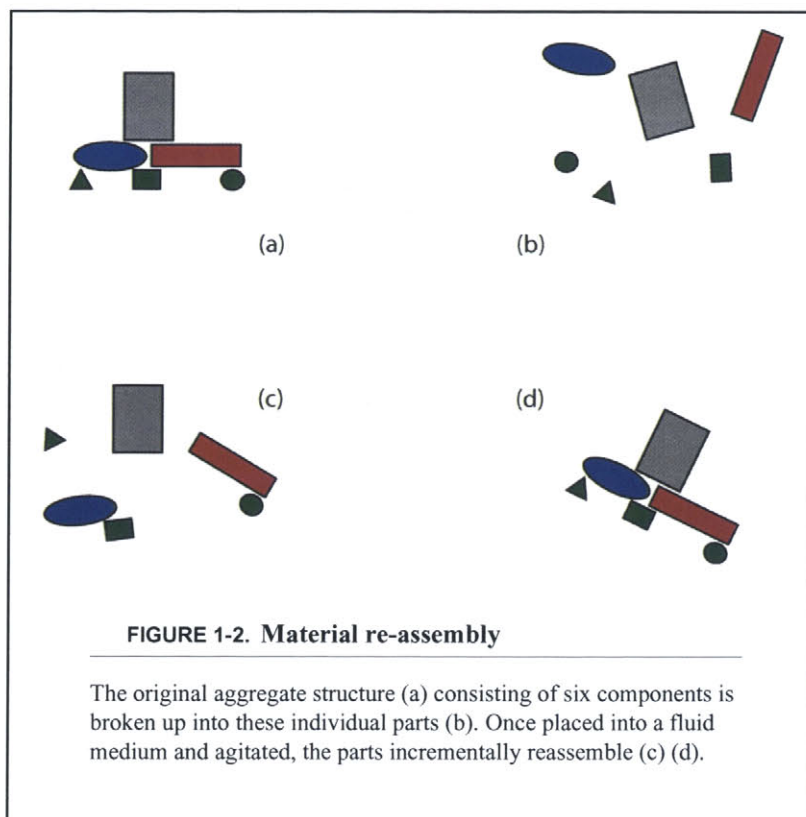
A loose mapping between material self-assembly and process self assembly follows basic intuition. The component parts of the material structure correspond to fragments of a running process[3]. The assembled material structure corresponds to the

_____

3. imagine the mapping the program and data space of a process onto a 2D memory, and then dicing that memory into irregular shapes. Each of these puzzle pieces would be a individual component in the self assembly.

original coded procedure represented as a 2D planar graph. And the fluid medium which supports the material self-assembly is the memory space of the machine on which the procedure is executed.

The mapping is necessarily vague, and different modes of material self-assembly will offer competing guidelines. It nevertheless points toward a powerful model for organizing computation. Processes which self assemble from mobile process fragments are potentially resilient to topological variations in the hardware, and should be robust to expansive changes in scale. With this behavior in mind, we state the thesis of this dissertation: **"A programming model employing a self-organizing ecology of mobile process fragments supports a variety of useful applications on a paintable computer"**

In support of this thesis statement, this work proffers several contributions:

- *process self-assembly*: a novel distributed programming methodology which maps existing techniques in material and virtual self-assembly to a broad class of dense ensembles of asynchronous, locally inter-networked computing nodes.

- a programming model built around a novel abstraction for inter-process communication, and the construct of a "process fragment" as the atomic element of process self assembly.

- illustrative examples of abstraction and modularity in process self-assembly.

- four applications that are both novel in their own right and that collectively demonstrate the *paintable*'s capacity for storage, communication and basic signal processing.

Roadmap

The remainder of this document lays out the rationale, builds the tools, explores basic concepts, extends them to applications and then takes stock. Chapter 2 motivates the architecture from an economic perspective, reviews background and related work, and specifies the criteria for evaluating the research. Chapter 3 presents the hardware reference platform, the programming model, and a simulator that embodies them both. Chapter 4 uses six simple examples to illustrate basic process fragment behavior and to introduce important programming constructs. Chapter 5 explores the application domain with four examples implemented on the simulator. For each application, the purpose and underlying algorithms are explained, and the functionality is demonstrated. Chapter 6 states the conclusions, lists the contributions and speculates on avenues for future work.

CHAPTER 2 *Background*

This chapter presents the background and motivation. A review of the economic fundamentals of IC chip manufacturing highlights the growing impetus for computing on sand-sized nodes. Work on Complex Adaptive Systems advances an alternate model of computing that is well suited to finely distributed architectures, the important caveat being that applications built on this alternate model have stubbornly resisted engineering refinement. Work on directed self-assembly of natural and artificial systems has produced techniques for guiding the self assembly toward bounded compliance with a predefined target structure. The chapter concludes with a list of criteria by which the design component of this research can be evaluated.

IC Economics

The search for techniques to program a *paintable* takes on an added urgency when you realize that you may soon need to use them. And a combination of feasibility and cost pressure make this a real possibility. The manufacturing capability for much of the *paintable* architecture will appear as a byproduct of the foreseeable advances in process technology. Indeed elements of the architecture are already in place (eg. micro and memory), while much of the remainder lies in the critical path of eagerly anticipated markets (eg. monolithic RF, active tagging).

Simple feasibility however is never sufficient to drive the deployment. Entrant technologies must always exhibit some compelling advantage over entrenched alternatives. For distributed computing on miniaturized nodes, the advantages include small size, portability and tolerance to harsh environments[1]. Beyond these limited specialty domains lie applications where finely distributed computing must contest directly with conventional architectures for use in mainstream information processing tasks. Here, one compelling advantage is cost, yes cost. Specifically, *dense ensembles of dust size computing elements support a price / MIP ratio with which centralized architectures can not compete, with potential for over 100-fold increases in compute capacity per unit cost.* The remainder of this section explains why.

For IC's sold en mass in a consumer market, the first order determinant of the price is the size of the unpackaged die. Even immense one-time expenses for initial development and plant ($\sim 10^9$ \$) can be amortized over monthly sales of several million units. In an industry characterized by a dizzying dynamic, the fixed cost of fabricating a single wafer ($\sim 10^2$ \$) has remained comparatively stable for decades. These fixed costs include manufacturing of the raw silicon ingot, slicing it into wafers, marching it through the ovens, packaging, and testing at multiple stages throughout the process. With the unit area of processed wafer as the fixed cost, the manufacturing cost for an individual IC depends on the percentage of the wafer's dies which are functional — the *yield* [2]. Alternatively, given a constant yield, the cost of a single IC depends the area of the die[3].

This simple relationship between yield, die size and unit cost have important consequences for economic viability of ultra-miniaturized computing nodes. A single defect is enough to render an entire die inoperable. By drastically minimizing the

---

1. an excellent example is the work on distributed sensor nodes. Centimeter scale nodes with power, computing, sending and communication are deployed ad-hoc and self organize to perform some coordinated sensing task. More detail in [50].

2. In this report, the term "yield" represents the percentage of a processed wafer that produces working dies.

3. Use of the die size as an estimate of manufacturing cost has a lower bound. For very small IC's, the minimum size can be defined by a pad ring — a rectangular ring of current drivers and contact pads for the bonding wires. And beneath a certain size, the cost of any IC is dominated by the packaging. However these lower bounds are seldom inviolate and often yield to technological advances in the face of economic pressure. For example, pin intensive IC's use ball grid arrays which, together with flip chip surface mounts have become an attractive alternative to explicit packaging.

economic penalty incurred by each defect, smaller dies enjoy a natural cost advantage over larger ones. How much? Consider the abstracted instance of three architectures; a high performance CPU measuring 1 cm^2, a smaller embedded microprocessor unit (MPU) measuring 25 mm^2, and an ultra small microcontroller core (MCU) measuring 1 mm^2. Contrived yet credible architectural features can be selected to render each of these architectures capable of equal amounts of compute capacity per unit area[4]. For example, at their nominal clock speeds;

$$1 \text{ CPU} = 4 \text{ MPU's} = 100 \text{ MCU's}$$

For defect-free manufacturing processes, these three architectures would deliver identical amounts of compute capacity per wafer. For processes where the yield is imperfect, the situation can be very different. Appendix A1 naively models process defects as point failures and steps through a coarse failure analysis based on three assumptions.

1. The likelihood of a failure at any given point is modelled as a 2D Poisson event

2. The presence of one failure within the boundary of a die is sufficient to classify the die as a reject.

3. Any one point failure will render one and only one die inoperable. Failures do not span boundaries to affect multiple dies.

Proceeding from these assumptions, we can calculate the expected value for the aggregate compute capacity recoverable from a wafer. Fig. 2-1.compares this data for the three architectures on a single log-log plot. The independent axis shows the total number of defects per wafer. At the extreme right, failures blanket the wafer and all dies of all sizes are inoperable (...bummer). On the extreme left, the yield is perfect and the expected compute capacity is maximized for all architectures (...it's Miller time). In the middle is a region where the sensitivity to defects varies dramatically depending on the die size.

Along the line labeled x_3 (corresponding to 1060 defects), 18,560 of the MCU's (95 % of the wafer) are still viable and available for generating revenue. Likewise, 201 MPU's (26 % yield) and 1 CPU's (0.5% yield) remain functional. So at the point where only one large die remains operable, there is almost a 200× difference in yield between the large and small format dies.

---

4. CPU → 1 GHz clock, 5 stage pipeline
   MPU → 250 MHz clock, 5 stage pipeline
   MCU → 50 MHz clock, no pipelining. no onboard cache
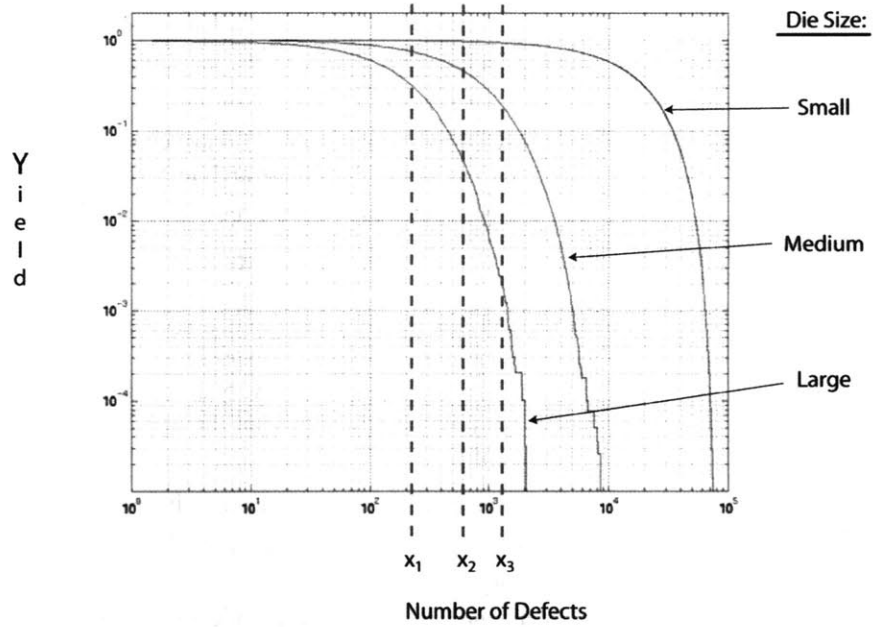
FIGURE 2-1. **Expected yield per wafer**

This log-log plot shows yield on a wafer as a function of defect count for three die sizes; 1 cm^2 (large), 25 mm^2 (medium), and 1 mm^2 (small). Statistics for three sample defect rates (labelled x_1 x_2 and x_3) are broken out and listed in table 2-1 below. Note that in the instance where 1060 failures leave one large die functional, the yield differential between the large and small dies approaches a factor of 200.

TABLE 2-1. **Yields for selected defect rates**

No. of Defects (label from plot of fig. 2-1)	Yield[a] (number of functioning dies)					
	Small dies		Medium dies		Large dies	
220 (x_1)	0.989	(19,380)	0.755	(592)	0.326	(64)
600 (x_2)	0.970	(19,006)	0.464	(364)	0.048	(9)
1060 (x_3)	0.947	(18,560)	0.256	(201)	0.005	(1)

a. The "yield" represents the fraction of the wafer occupied by functioning dies.

Process yields are among the most tightly guarded secrets in the IC industry. Yet products are commonly produced from processes operating in a yield regime of 30 %. And while a given process is often matured to support a much higher yield, the motivation can be undercut by the introduction of a succeeding IC process[5].

The take away message from this section is; *measured in raw compute capacity on silicon IC's, the large form factor, "high performance" dies are the among most expensive, inefficient form that computing can take.* And that the only reason that we tolerate this surcharge is because we lack robust techniques for efficiently distributing our computing over numerous fine grain ensembles — an issue that is taken up in the next two sections.

Complex Adaptive Systems.

Of course, the difficulty with the preceding analysis is that it treats raw compute capacity as a universal currency. Unfortunately, in the prevailing models of computation, not all ALU clock ticks are equal. Most computations that are expressed procedurally as a sequence of instructions on a Turing equivalent machine do not distribute efficiently, if they distribute at all. Comparisons based solely on compute capacity are inherently flawed in that they imply that a procedure runs on both architectures and produces identical results. In instances when this requirement is enforced, the cost advantage of the miniaturized nodes is all but lost in a sea of overhead for control and synchronization.

This section examines the work on Complex Adaptive Systems as a vehicle for recasting the concept of "computation" into a form that is more amenable to computing on *paintable* class architecture.

Design of contemporary computing systems has been constrained by a number of seemingly reasonable assumptions about architectural reliability and performance criteria of both the hardware and software. For example, few people are interested in a sorting procedure that, for a dramatic decrease in run time, will produce results that, while imperfect, are still "good enough". The difficulty in this case is the definition of "good enough". Similarly, given a procedural description of a task, few

---

5. consider the effect that bringing up a 0.1 micron process has on an existing 0.18 micron process. A 20% yield on the 0.1 micron line is equivalent to an 65% yield on the 0.18 line.

people are interested in ultra-cheap yet faulty hardware that will get most of the job done most of the time. Computing was defined a series of discrete state transformations executed under direction of coded instructions assembled into a procedure. The only suitable computing elements were those that could read the instructions from the stream and then faithfully sequence through the associated computational events. The only suitable computations were those that could be uniquely described by a procedure. System designers internalized these assumptions into a mindset and set of axioms that guided the development of most man-made computing systems.

Scientists investigating the dynamics of natural phenomena have developed an alternative view of computation which has challenged elements of the computer designer's canon. Work on natural systems ranging from neurons to weather patterns has produced several key insights:

- these systems could be modeled to some level of fidelity by computational models. To the degree that this worked, the natural system could be regarded as 'computing' — and doing so using 'hardware' that was heretofore regarded as unsuitable. It's not a computer, but it's computing!

- when comparing various models, investigators had to rethink the goals of the given computation in view of the goals of the entire system. For example, in time critical applications, doing half the job in a fourth the time may be preferable to running to completion, particularly if errors could be cleaned up downstream.

- the development of the computational models was at least indirectly informed by observed characteristics of the underlying hardware — ants, neurons, particle systems, Wall Street traders — all very non-traditional hardware.

An early example of this approach was David Marr's work on vision in the 1970's [32]. Departing from mantras of the time, he cast vision as a computational task and sought to characterize the computation on three levels: the *competence* (objective), the *algorithm* and the *hardware*. His approach was to hypothesize the objective of a given computation (building a visual precept) in the context of the goals of the larger system (survival). This first step was a precursor to any consideration of the algorithm and form of the hardware.

This and subsequent work on "Natural Computation" [4] broadened our formal definition of computation by establishing new metrics to measure the efficacy of computing devices and algorithms. Returning to the previous example, it is now acceptable to consider an imperfect sorting algorithm because the bounded algorithmic shortcomings could be balanced against speed of execution, size of the code, suitability for novel hardware and the ability to make up for inevitable errors

later. Similarly hardware with bounded inconsistencies in performance can be seen as preferable if it is uniquely suited to a harsh environment, adaptive in some task specific way and is resilient to failures of its component parts.

This and related work on Complex Adaptive Systems provide the underpinnings for an alternative form of computing that is central to this research.

- Arbitrarily complex system behavior can be created from large numbers of simple processing elements.
- Reliable computation can be expressed as the aggregate statistics taken over a large set of local interactions. Dependence on statistics decouples the global result from the outcome of any one local interaction.

Work on StarLogo [43] captures the strengths and the pitfalls of this approach. Autonomous virtual creatures, coded with simple behaviors, interact to mimic the global behavior of slime mold, traffic jams and forrest fires. Using StarLogo as an experimental platform, even high school age children can synthesize complex phenomona. The drawback is that no insights emerge as to methods for engineering the second order behavior. Programming is largely trial and error.

Self Assembly

Self-assembly appears in several forms, each distinguished by the complexity of the structures that they generate and the ease with which one can characterize (and replicate) the behavior.

Scaffolded self-assembly. Objects position themselves into shape complimentary receptacles. Any lock-and-key scheme qualifies as scaffolded self-assembly. A good example is the Fluidic Self-assembly technique from Alien Technology [1]. Small objects are cut to one of several different shapes and slurried over a surface. The surface is patterned with depressions whose shape matches one of the shape types of the objects. Objects in the slurry which contact a depression with matching shape tend to lock into place.

Thermodynamic self-assembly. Also called entropic self-assembly. Objects move under the influence of attractive and repulsive forces exerted on them by other objects. The net force exerted on an object corresponds to the object's instantaneous free energy. The aggregate structure corresponds to the configuration with the minimum free energy (at least a local minimum).

An oft cited example from nature is the water droplet. The repulsive forces exerted on water by the air favor shapes with the minimum exposed surface area. An early example from the regime of man made systems is the smooth flat beds of molten metal used in glass manufacture.

Coded self-assembly. This is the most complex form. Individual interactions are guided by coded instructions embedded in the agencies. An agency's selection of code is a function of the local environmental and the agency's instantaneous state. While the interactions are local, the codes which direct them can be arbitrarily complex.

This is the form of self-assembly most commonly employed in the coordinated activity of large ensembles of robots. Coded self assembly has the potential for managing complexity on a scale comparable to that of living systems. However it is the most difficult to characterize and emulate.

Systems which self-assemble offer a rich set of insights and techniques for use in information processing. The trick has been to map the behavior from the material domain to the virtual. Several research projects have taken up this challenge.

An early programming application of scaffolded self assembly was Hewitt's Planner[26]. Planner employed *pattern directed invocation* to dynamically assemble a process from a set of predefined software components. The hardware venue was a traditional single processor architecture. The atomic components were algorithms that were packaged for autonomous execution and labeled externally by patterned "keys". An executing component in need of a particular function would express the function as a key and broadcast a request. Available components with a matching key were candidates for servicing that request, with control passed to the selected candidate.

The work of the Amorphous Computing Group produced the first instances of coded self assembly targeted to domain characteristic hardware[1]. They defined the target hardware as a dense ensemble of randomly positioned miniaturized processing nodes, each running asynchronously and communicating locally. In the spirit of bulk commodities, individual particles were subject to sporadic failure, both individually in large groups.
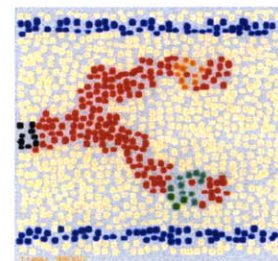
In their original programming model, all code is permanently embedded in the particles at the time of manufacture. This code contains multiple functions that can be "woken up" in response to a number of predefined conditions. With the code statically positioned, the principal manifestation of the self-assembly is the assignment of property labels for the particles. While these labels can correspond to patterns of arbitrary complexity, they are all created by unsupervised local interaction, and each exhibits acceptable fidelity to a predefined target pattern.

Coore developed the concept of a "growing point" as a tokenized interrupt that used pheromones and tropisms[6] to direct its migration among the particles [14]. Nagpal extended the hardware model to approximate the mechanical properties of epithelial cells [40]. Radiating gradients carried messages and built estimates of the distance from dynamically selected reference points. Messages were combined with barrier synchronization to sequence the assignment of material properties and propagate new gradients.

The goal of this research is to extend virtual self-assembly to include fragments of a coded procedure as the atomic element of the assembly process.

Criteria for Success

Ultimately, the success of this research is measured by the support it lends to the thesis statement. However, lurking behind this thesis statement were two larger agendas: 1) to provide a crucial enabling technology to an emerging architectural class, and 2) to demonstrate the class's practical importance. The conduct of this work necessarily involved a substantial design component for each of three core elements: a hardware reference platform to define the computing environment, a programming model that engendered self-assembly, and a set of applications to qualify the architecture's utility. Throughout, design decisions were made, often in the face of numerous alternatives.

As an additional vehicle for evaluating this research, metrics are presented for each of these three components. Some of these metrics can be expressed in checklist

---

6. This work makes heavy use of biological metaphors. In this case, "pheromones" are gradient fields with embedded state information. A "tropism" is a function of several gradient fields.

form and evaluated in a yes/no fashion. Others resist formal reduction and have to be argued rhetorically. Here is a breakdown of the questions that must be answered.

Hardware Reference Platform. As an architectural blueprint, the questions to be asked about the reference platform are:

1. How well does the reference model capture the fundamentals of an architectural class? Is it overdefined, underdefined? Does it capture the characteristics of interest without being unnecessarily restrictive?

2. Is it sane? Do any of the assumptions offend basic laws of physics or economics? Physical laws to watch are power consumption and communication capabilities. Economic "laws" relate the utility of the *paintable* to that of the more entrenched architectures such as classic embedded controllers. Are there foreseeable circumstances likely to obviate the *paintable* as an architectural class?

Programming Model: The specific metrics of interest here are:

1. Does the programming model adequately define its underlying abstractions, program components, and basic computing resources available in the particles?

2. Does the programming model support the fundamentals of self assembly? No claim is made to understand <u>the</u> fundamentals of self assembly. Rather, we are looking for faithful incorporation of a well articulated subset.

3. Does the programming model support data exchange among the assembled program components?

4. Does the programming model compromise any of the affordances of a *paintable*? Does it have difficulty scaling when the number of particles spans several orders of magnitude?

Applications: There are many flavors of self assembly. Did we chose the right one? The answer should be evident in the applications. It is up to the applications to collectively demonstrate that the architecture supports performance sufficient to merit further attention. The evaluation criteria here are:

1. Are the applications "real"? Namely, are they functions that people currently use, could possibly use or that solve existing problems?

2. Do they make sensible use of the architecture? A handheld calculator running on a 250 GOP (giga-operations per second) machine doesn't count.

3. Is the performance robust under the variability inherent in a *paintable* (unpredictable topology, intermittent failures, expansive swings in the size of the ensemble, etc.)?

4. Is the specific application demonstrative of a broad class of algorithms?

5. Do the applications actually employ self assembly. Any procedure which could execute its entire function on a single particle or with many copies of a single process fragment is uninteresting.

Throughout remainder of this report, we will revisit these questions.

Onward ...

System Architecture:

Programmable Particles, Computing Substrate

This chapter is the fine print — a lot of it. It presents a reference platform for the hardware, a detailed model for the programming, and a simulator that embodies them both. The centerpiece is the programming model. Novel abstractions for memory usage, inter-process communication and component interaction define a versatile new model for concurrent computation. The hardware reference platform describes the underlying computing environment with a model of a particle's internal architecture. Various component implementations are examined and subsumed into a single abstraction based on a universal machine fetching code and data from local memory while exchanging messages with networked neighbors via error-free channels. The concept of pushpin computing is outlined as a contemporary illustration of this architectural class. Finally, a dedicated simulator supports a graphical programming environment for development and test of software for the succeeding chapters.

This chapter is the lion's share of the research. It is voluminous and necessarily dense. By the end, you will know what you are programming and how to program it, but not why. The jumble of rules, abstractions and nomenclature will be consistent and often accordant with the metaphor of material assembly. Yet, it will take the next three chapters to ground them and motivate their use. Still, every good game has its rules, and the fun doesn't start until the rules have been read.

Hardware Reference Platform

As a preamble to defining a modular programming model, this section describes a hardware reference platform that defines the computing environment inside the particles. The crucial attributes of this platform are that it is overtly conventional, universally programmable and tightly resource bound. Modest RAM storage, organized in a linear address space, contains executables that run on a single general purpose computing element. All external I/O is managed through a single networking subsystem which supports the abstraction of error-free data exchange with a small number of peripherals — in this case, spatially neighboring particles represented as virtual portals (fig 3-1). Four properties delineate this architectural class:

1. All addressable neighbors are spatially proximal to the particle — with the neighborhood defined by the communication range of the network subsystem.

2. The number of addressable neighbors (N) can vary unpredictably, reflecting the fact that the number of neighboring particles within a communication range can vary; either intentionally or unintentionally. The network subsystem must respond by automatically adjusting the number of virtual peripherals ports that it maintains[1].

3. Messages sent to the neighbors can exhibit probabilistic transit times and are not automatically acknowledged. While verification can always be explicitly requested, all messages have an unconstrained latency. Transit time (t) can be described by a probability density function p(t) which, after some multi-modal transients, asymptotically approaches zero as $t \to \infty$. Conversely, messages that are received are assumed to be free of error[2].

4. All code and data which is not prestored in a particle must come from neighboring particles — or other devices communicating under the guise of a particle. Any external device can exchange data with neighboring particles by adopting the particle's wireless protocols and mimicking its network behavior.

For the remainder of this report, *we adopt the platform definition above as a sufficient description of candidate hardware. Any particle designed to this loose specification will be amenable to the tools and techniques of the succeeding text.* By adopting this generic superset of the paintable as the reference platform, we side-

---

1. It is up to the network subsystem to maintain the mapping between the virtual port that represents each neighbor, and the actual neighboring particle.

2. This delegates to the network subsystem the task of error detection, error correction and, as necessary, request for retransmit.
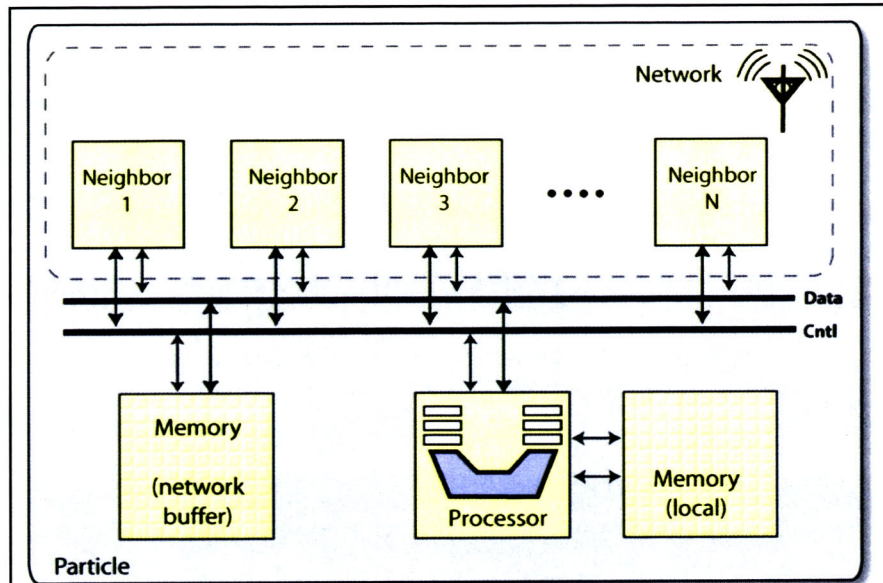
FIGURE 3-1. Hardware Reference Platform

The hardware reference platform maps the messy physicality of deeply embedded systems to the sterile virtual constructs that underlie coded procedures. Software modules execute on a standard fetch-decode-write computing element operating on local memory. The processor can also exchange messages with the neighboring particles, each represented locally by homogeneous virtual ports created and maintained by the network subsystem. The hardware supports Direct Memory Access (DMA) transfer between a port and the local network buffer.

The tight coupling between the physical world and the embedded particles is reflected in the nature of these virtual ports. They can appear and disappear unpredictably. However the number of virtual ports always reflects the number of actual particles with which the network subsystem maintains active contact. Transit time for the message is probabilistic, with a vanishing but nonzero likelihood that the message will fail to arrive. Particles can only communicate with other particles. External devices seeking to exchange data with a particle must mimic the wireless behavior of a particle in order to gain surreptitious access to a local neighborhood.

The reference platform makes no explicit mention of the powering subsystem because the choice of power source does not normatively impact the structuring of the software components — at least not directly. Indirectly, the support for pseudo-random positioning of the particles produces the variability in the size of a particle's neighborhood. — an effect that the networking subsystem must compensate for.

step dependence on the engineering and economics of speculative subsystems for wireless networking and power harvesting, and admit an extended array of form factors and applications — from millimeter scale paintchips and laminar particles, to centimeter scale pushpins tacked up on a wall, to decimeter scale nodes air-dropped over open terrain, to meter scale buoys clustered on the open sea.

The remainder of this section expands on this platform definition with reviews of individual subsystems[3], and a look at pushpin computing as an initial solution to the problems of power and networking. While stopping well short of a complete design, our treatment of the pushpin machines suffices for two crucial sanity checks:

1. an implementation of the reference platform can be built using commercially available components and materials

2. at least one implementation of the reference platform supports a minimum risk path for monolithic integration into the 4 mm^2 size regime (i.e. is easy to integrate into a single low cost IC).

Component Subsystems

Processor/Memory. Each particle is fitted with a microprocessor which serves as the particle's universal computing element. All executable code resides in the particle's RAM and executes on the microprocessor. The instruction set must be rich enough to efficiently execute any instruction stream compiled down from a general purpose programming language such as C. Minimum RAM size is 50K words organized in chunks of 8 bits or greater spread over a 16 bit address space. While not required, hardware assist for interrupt scheduling and for integer and floating point math is probably a good investment. The internally generated clock should support an operating range of 10-200 MHz, dynamically adjustable to account for available power and network bandwidth.

This definition admits many variations on common architectural themes; vanilla von Neuman CISC cores, Harvard architecture, super scalar, RISC, and VLIW. However, given the emphasis on power efficiency, the simpler bare bones models

---
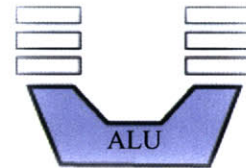
3. Computing elements, networking and power.

may be preferable. Micros in this class are already manufacturable in sub 1mm^2 form factors[4]. Additional work remains to minimize the power consumption.

Power. There is nothing to preclude a particle from operating exclusively from battery storage. But most applications involving statically positioned particle ensembles will want to take advantage of harvestable power available in the immediate environment. There are almost as many power harvesting techniques as there are environments to harvest from; chemical, mechanical, optical, and electro-magnetic coupling. Virtually any technique which produces an electrical potential across two points is of interest, provided that it affords the particle at least some latitude in its positioning — namely no precision placement and no dedicated interconnects.
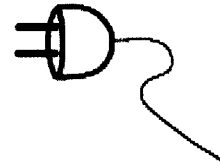
Some example scenarios:

- photovoltaics: solar cells fitted to one or more sides of the package.
- chemical: (primitive battery) for centimeter scale packages distributed over a surface with the correct chemicals. Or for very small packages thrown into a vessel containing the correct chemicals (think stomach).
- structural: pins protruding from the particles pierce layered membranes to draw power from isolated planes. More detail is given below in the text on pushpin computing.

For most techniques of interest, the amount of available power will fluctuate depending on the particle's position, particle density and the material characteristics of the immediate environment. Consequently, the other subsystems must be able to match their consumption to the available power[5].

In the selection of any power harvesting technique, the two dominant questions are: How much can we get? What is the minimum that we need? The amount of available power will vary greatly depending on the technique under consideration. The amount of power consumed will typically be dominated by the needs of the network subsystem[6].

---

4. Consider the 1.2M transistors of Intel's venerable i486. A geometry-only shrink to 0.1 micron is sufficient to confine it to a 1 mm^2 area.
5. Possibly varying the frequency of the clock or by cycling through wake/sleep modes.

Networking. Data exchange between a particle and its environment is gated through the particle's network subsystem — a wireless transceiver that supports a minimum average particle-to-particle bandwidth of 100 kbs full duplex. As with the definition of the power subsystem, the term "wireless" does not mean contactless. Rather it implies "directionless" with a strict prohibition against precision hardwired interconnects that would constrain the positioning of the particles.
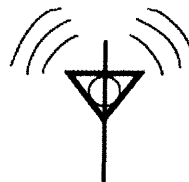
The reception area should be at least approximately circular, with a radius that yields an average neighborhood size of 8 or more particles. Within this radius, the network subsystem performs basic link management, periodically running its node discovery and verification functions to maintain an enumerated list of the neighboring particles with which it can speak. Changes in the neighborhood particle topology, both intended and unintended, are automatically reflected in this list. Channel coding and error detection/correction are employed as needed to maintain signal integrity and support the abstraction of a dedicated, error-free link.

Communication is strictly peer to peer with no inherent hierarchy among locally communicating peers[7]. As an example from contemporary practice, consider ethernet based LANs. All nodes connected to the medium (usually coaxial cable) are considered part of the local network. Signalling is one-to-many with one node transmitting while the remainder listen. Speaker assignment is contention based with instantaneous bandwidth relying on the statistics of random collisions. Mapping this analogy to an ensemble of paintable particles, the network connection among the P particles in an ensemble can be regarded as the aggregate of P overlapping mini-networks. Each mini-network is centered about one particle and is defined to consist of that particle and the neighbors with which it can communicate. Each particle defines at most one mini-network. But a particle with N neighbors will simultaneously be a member of up to N+1 distinct mini-networks.

While the term "wireless transceiver" naturally suggests near-field RF as the underlying link, there are a number of basic problems which motivate a look at alternatives.

---

6. The expectation is that 1mW will suffice for the micro.

7. A hierarchy can emerge from networking software layered onto the underlying communication link. However no hierarchical organization is implied by the hardware. This approach contrasts with Bluetooth style pico-nets and other standard cell-based networks.

- **luminescence**: while still a research topic, progress has been reported on efficient light-emitting structures which can be patterned directly onto CMOS dies [3]. Particles fitted with these light sources and omnidirectional photosensors could communicate through a physical medium whose translucency was matched to the desired communication radius. Motivation for this approach is the copious bandwidth native to optical signalling.

- **electrostatics**: for particles in the size regime of 10-200 mm^2, two electrostatic techniques —capacitive coupling[34] and dissipative loading — show initial promise. While the bandwidth is necessarily limited, the technology thresholds are modest enough to permit development using commercially available components.

- **near-field RF**: The recent past has witnessed spirited commercial interest in miniaturized RF transceivers, with a number of monolithic devices either already announced or in late stage development. The breadth and depth of this interest is sufficient to ensure a steady rate of progress in the bandwidth, footprint and power efficiency of mobile RF systems for at least the next years.

However, use of RF in particles presents several basic difficulties, depending on the scale. For use with particles on the size scale of several cm^2 operating in open environments, the severe anisotropy of the reception area negates an assumption crucial to results in the following chapters. For particles on the sub 10 mm^2 size scale, the antenna design and the supra-GHz carrier frequencies represent the kind of obstacles that necessitate more basic research.

Pushpin Computers

As a vehicle for characterizing the basic architecture, feasibility experiments were conducted on designs for pushpin computers. In the pushpin variant, particles are sized to the cm^2 scale, communication is based on electrostatic sensing, and power is delivered through catheterized pins that protrude from the package to make mechanical contact to rubbery conductive planes. While construction of a truly paintable IC would have involved protracted development at the component level, pushpin computers enjoy a relatively modest hardware threshold, and yet are functionally compliant with the reference platform.

As a precursor to a wider treatment by Lifton[34], initial experiments were performed on a 3-pin particle. Here, the computing elements are targeted toward bean sized packages with the three pins protruding from the base. The pins are of unequal length, catheterized so that only the tips make electrical contact. These 3-
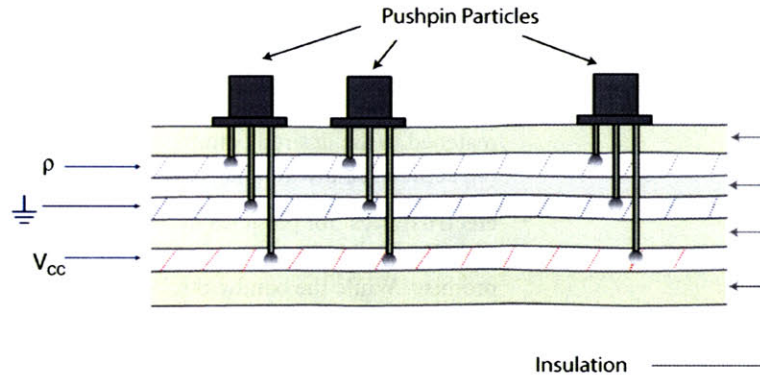
FIGURE 3-2. Pushpin computers positioned in layered composite

A cross sectional view of composite populated with pushpin computers.
3-prong computing elements (pushpin computers) are thumb tacked into layered composite carrying supply voltage, ground, and a resistive layer used for signalling.

prong pushpins are thumb tacked into a planar composite consisting of seven sheets of soft rubbery silicone (fig. 3-2). Two of these layers are electrically conductive, separated by intervening layers that are non-conducting. The upper most active layer is resistive, with ohmic rolloff naturally limiting the range of any current-driven signal radiating from a point source. In operation, pushpins are randomly positioned on the composite's surface and use the two longest pins to contact the conducting planes to draw power. The third pin contacts the resistive sheet and alternates between radiating a signal and listening for an answer.

Fig. 3-3 shows a lumped parameter model for communication via a resistive sheet. The effective resistance between the communication pin of a transmitting particle to the ground pin of a receiving particle is modeled as the cascade of an effective sheet resistance R_{mi} and a fixed shunt resistance internal to the particle R_s. In the talk mode, a particle passes a bandlimited binary stream through the communication pin. Neighboring particles in listening mode sample the voltage drop across the shunt resistor using a comparator and a uniform threshold V_{th} to recover a binary stream. The effective resistance of the sheet increases with distance, essentially limiting the distance over which a logical high can be sensed.

FIGURE 3-3. Pushpin Network

A pushpin uses one pin to contact the resistive sheet. This communication pin is temporally multiplexed between a talk mode and listen mode. Particles in the talk mode pass a bandlimited binary stream directly through the pin into the sheet. The sheet acts as a dissipative channel connecting the transmitting particle to multiple listeners, with the amount of ohmic dissipation increasing with distance. Listeners sample the signal level using a thresholded comparator to recover a binary stream.

Three system characteristics crucial to any paintable networking method are the communication radii, the signalling bandwidth and the power dissipation. For the dissipative pushpin network, these performance figures were sampled using the test apparatus of fig. 3-4. A sheet of dissipative anti-static polymer, measuring 6 x 11 inches, was fitted with a set of 330 sockets that were positioned irregularly and bonded using conductive epoxy. Transmitting particles were modeled by three unpopulated sockets onto which a signal could be externally applied. Receiving particles were represented by sockets populated with discrete resistors whose complimentary leads were connected to a ground plane.

For a given point source, the size of the reception area is a non-linear function of three variables; the spatial density of the receivers, the absolute volume resistance of the unpopulated sheet[8] (R_v) and the ratio of the sheet volume resistance to the shunt resistance (R_v/R_s). Fig. 3-5 plots the signal strength as a function of distance for two values of (R_v/R_s). Data was collected by applying 5 volts DC to an unpopulated socket. For each of the populated sockets, the induced voltage was measured

---

8. R_v is estimated by averaging the point-to-point resistance between sample point pairs on an unpopulated sheet.

FIGURE 3-4. Epoxied Resistor Jungle (ERJ)

Top: Front view of polymer sheet mounted in a wooden frame. 330 sockets are randomly positioned on the polymer and most populated with shunt resistors. The shunt reistors extend upward from the sockets to contact a conductive mesh that is tied to ground. All but three of the sockets are populated.

Left: Rear view of polymer with the base of the sockets exposed for electrical measurment. The bases of the unpopulated sockets are circles.

Right: close up of top showing one of the unpopulated socket.

at the base[9] along with the straight line distance back to the "transmitting" socket. This procedure was repeated for each of the three unpopulated sockets. Fig. 3-5 shows the sampled data points and a fitted curve. Note that for a given particle density, selection of a threshold voltage is sufficient to define the expected size of a neighborhood.
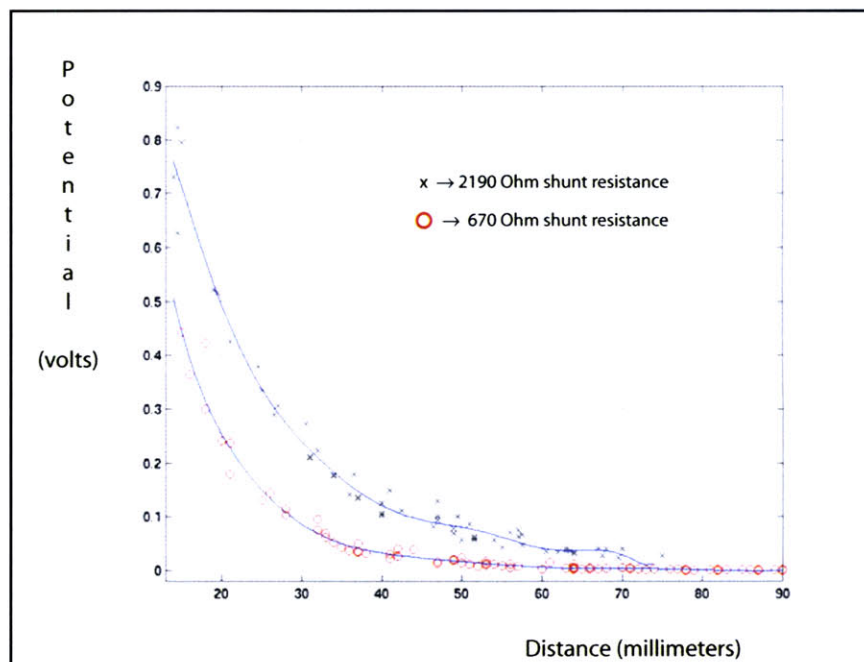
FIGURE 3-5. Distance dependent attenuation in the ERJ

A 5V DC potential is applied at the unpopulated sockets on the ERJ. The induced potential is measured on the populated sockets and plotted as a function of distance. The test is conducted for two values for the shunt resistance (R_s in fig. 3-3). In the design of a network, selection of a threshold voltage defines the size of the neighborhood.

---

9. The point at which the shunt resister contacted the socket.

Discussion. Nothing in this brief treatment approaches closure on the many crucial implementation questions surrounding the hardware for pushpin computing. No data was gathered on the failure modes of the conducting silicone used in the power planes. And in the networking subsystem, no consideration was given to the additional power consumption or the bandwidth costs for clock recovery, channel coding or error management. These challenges and more have been taken up by Lifton[34].

The crucial message of this section is that the relevance of the programming model developed herein is not confined to a specific design for pinless paintchips with their speculative subsystems for power and communication. Rather, the programming model can be designed to an extended architectural class — one that includes tractable designs based on conventional components. Joint development of the hardware and software for ultra dense distributed computing can begin with manageable hardware that captures the essentials of the problem domain.

Programming Model

To program a *paintable* is to program on both sides of a great divide. On the one side, programs must embody an organizing principle that anticipates the irregularity, asynchrony, fluctuation, expansive scale, and complex behavior of the particle collective. On the other side, programs must necessarily consist of modules which run on individual particles where the computing environment is well ordered, consistent, and reliable. This section presents a programming model (PM) for the *paintable*, and illustrates its application with a simple run-time example. The mechanism for extra-ensemble data exchange is revisited. Work on related programming models is reviewed.

But first, whence the metaphor of material self assembly?

Computation is often applied to model events in the physical world. A successful approach has been to pattern the structure of programs after the structure of physical events that they model. *Object oriented programming* organizes programs into 'objects' whose instantaneous state is internalized as a collection of state variables and whose exposed interface is a set of object-specific procedures. *Stream programming* uses delayed execution as a model for the progress of information through cascaded physical systems.

As computing migrates onto densely distributed embedded substrates, a programming abstraction based on material self-assembly[10] follows naturally. As with the *paintable*, the "hardware" in material self assembly consists of numerous, unreliable components arranged in a heterogeneous ensemble whose native complexity strains traditional analytic techniques. Material self-assembly, when viewed as a computation, offers fresh approaches toward exercising quantitative control over these seemingly chaotic systems. Finally, a programming model based on material self-assembly captures the dichotomy inherent in computing on a *paintable*. Simple behaviors guiding local interaction must be engineered to yield global behavior which is complex, adaptive, yet to at least a degree, predictable.

---

10. "self-assembly" is broadly defined to include several common variants (scaffolded, entropic, and coded) operating on a variety of substrates (biological cells, insect colonies, man made nano-structures).

PM Specification

The programming model is built on three cornerstones; the process fragments, the shared memory partitions, and the embedded operating system (OS). Process fragments are autonomous, self contained computational elements that interact locally to perform coordinated tasks. Shared memory abstracts both the inter-process communication and the inter-particle transfer. The OS regulates the operating environment, schedules local resources and supports the process fragments with a collection of service functions packaged as a "toolkit".

Once the programming model is in place, this text concludes by revisiting the model for I/O with the external environment. Additional detail is given on the required similarity between an I/O portal and a particle. The format and methods of data exchange are defined.

Particle Memory Organization

Software on a *paintable* is organized into autonomous, self-contained executables referred to herein as "process fragments" (pfrags). All pfrags running on the particle's micro reside in the particle's RAM space. Most of the RAM is available for pfrag program and data. However a section of the RAM is reserved what is called the I/O space an area which is at least readable by any pfrag running on the particle's micro (fig. 3-6).

A subset of the I/O space is called the HomePage. The HomePage is an area where pfrags can both read and write tagged data. Any pfrag local to the particle can post to the HomePage. And posts to the HomePage are readable by all local pfrags.

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

The basic unit of information on these bulletin boards is the post — an instance of data that is tagged for identification. Posts are structured as key/value pairs, where the value component can be another key/value pair. The size of the posts is regulated by the particle's OS and is bounded by the space available on the HomePage.

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
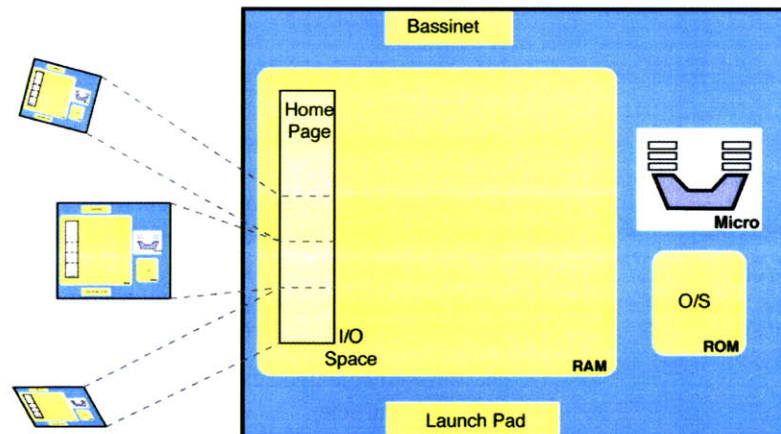
FIGURE 3-6. Organization of Particle's RAM

> Most of the RAM is used as executable space for the currently running programs. However the IO space is reserved. A subset of the IO space — the HomePage — is available for programs to read and write tagged data. The remainder of the IO space is read-only.

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[11] triggers an adjustment in the number of mirror sites.

Actual dimensions will necessarily be technology and task dependent. As an illustration, the simulations of the following chapters assumes a total particle RAM capacity of 50K words. Each HomePage is allotted 1K words. Each particle has an average of 15 neighbors, yielding an average I/O space of 16K words — roughly a third of the total RAM space.

Finally, two additional FIFO's are defined that support inter-particle transfer of the process fragments. The Bassinet is an input FIFO where incoming process frag-

---

11. Due to either a run time fault or the intentional appearance / disappearance of a particle.

35

ments are assembled during serial input. Similarly, the Launch Pad is a staging area where outgoing process fragments are buffered for streaming out to one of the neighboring particles.

Process Fragments

The atomic element of the *paintable* programming model is the process fragment (pfrag). These are autonomous program entities that migrate among the particles and interact with the local environment. As a subclass of general computational processes, pfrags consist of coded behaviors and state. The state is internalized as data in a vector and the behaviors are defined by sequences of machine instructions. The additional restrictions that distinguish process fragments as a subclass can be reduced to three normative requirements:

1. They are self contained executables capable of fitting entirely in the RAM space of a single particle.

2. They gate their entire I/O through the I/O space in the particle's RAM, with writes directed to the HomePage and reads taken from anywhere in the I/O space.

3. They define behavior for 5 functions which the particle's OS can issue to them at any time. These functions are public handles similar to the public methods of traditional Object Oriented Languages such as Java.

Sized to fit Conceptually, *paintable* process fragments are intended as self contained executables. In the strict sense, this means that no pfrag can allocate scratch memory or execute a branch instruction that points to an address outside the fragment's bounded address space. However, implementation has motivated three exceptions:

- A process fragment can request from the OS a parcel of scratchpad memory. This is a one time request that is made as part of the fragment's initial negotiations for entry into the particle. This memory is exclusively bound to the process fragment and is automatically freed when the fragment departs or is erased.

- The OS can also maintain a collection of library functions that it makes available to pfrags as needed[12]. The flip side is that a pfrag must be prepared for an instance when a requested function is not available.

---

12. See description of Pfrag Toolkit on page 42.

- Finally, while it is outside the spirit of this spec, there is nothing to stop a pfrag from using the HomePage as temporary data storage[13].

Gated I/O All data traffic between a process fragment and its environment is gated through the I/O space and managed by calls to the OS. Data on the I/O space are organized as posts — variable length packets structured as key/value pairs.

Process fragments use OS function calls to query the number of neighboring HomePages that have been mirrored in the I/O space, to inquire as to the number of posts on a given HomePage mirror, to sense the available free space on the local HomePage, and to read / write the posts. These OS functions are limited to memory management. The OS has no capacity to interpret the keys in the posts or to pre-process the posts (i.e. no searching, sorting, or filtering). Pfrags must also be prepared for the case when lack of space on the HomePage causes posts to the HomePage to fail.

A process fragment receives the posts as a string of data with a reported length. It is up to the pfrag to interpret the keys in the post in order to assign meaning to the remaining data. There is no requirement that all posts be meaningful to all pfrags. Indeed, most posts will be indecipherable to most pfrags. The alternative would necessitate some sort of centrally maintained global semantic taxonomy — of potentially unbounded size. In the implementation of the following chapters, we circumvent this problem by limiting the key values to pfrag tags. These tags uniquely identify pfrag types and are assigned to the pfrag during authoring from a centrally maintained database accessible from the development environment.

Process fragments do not communicate directly. Rather, they infer the type and state of other local pfrags from the posts in the HomePage. Currently, pfrags that wish to communicate with each other must do so by passing tagged posts back and forth through the HomePage. Work on the applications highlighted the shortcomings of this approach for data exchange. Future revisions of this spec will provide an alternative, probably involving message passing via the HomePage to set up DMA-style transfers between the pfrags.

Public Handles The process fragments' behavior is completely defined by their collective response to 5 required handles: *Install*, *DeInstall*, *Transfer-Refused*, *Transfer-Granted* and *Update*

---

13. yet

(state vector)

Install
De-install
Transfer Granted
Transfer Refused

(public handles)

Update

(private routines)

(local scratch)
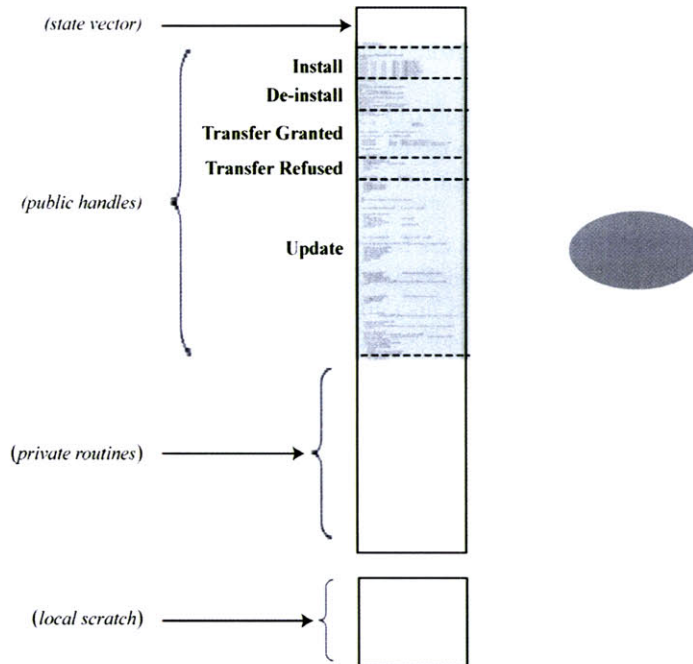
FIGURE 3-7. Process Fragment

Process Fragments are self contained executables with public handles for 5 required functions. In addition to these functions, the pfrags must carry as payload sufficient scratch space for any internal calculations and any necessary private routines. The exception to this being those subroutines explicitly supported by the OS.

Except in rare instances, four of these handles are comparatively simple. The *Install* and *DeInstall* handles carry out the pfrag's portion of the housekeeping involved with positioning the executable in memory, running it, stopping it and removing it upon completion. The *Transfer-Granted* and *Transfer-Refused* handles are the pfrag's portion of the handshaking involved with the migration of the pfrag from particle to particle. The lion's share of the behavior is embodied in the *Update* routine, which is intermittently called by the embedded OS. These five commands are itemized in table 3-1 and discussed in more detail below:

Install On successful transfer into a particle, a pfrag is assigned a starting address in the RAM by the OS. The OS then copies the fragment out of the bassinet and signals it to complete its part of the installation with a one time call to the fragment's *Install* routine. Typically a pfrag places an initial post into the HomePage and runs some internal initializations. If in the course of negotiating its transfer, a pfrag requested scratch memory, that memory is also assigned during the *Install*.

DeInstall The complement to the *Install* handle is the *DeInstall* handle. If *DeInstall* is called with no parameters, the pfrag must immediately begin to erase any of its posts on the HomePage, de-queue any transfer requests and finally mark itself for erasure by the OS. The *DeInstall* can also be called with a single parameter, which is interpreted as a grace period during which the pfrag can secure permission to transfer to a neighbor. If the grace period expires and the pfrag has not exited, the OS will typically issue a *DeInstall* with no parameters.

Transfer-Granted Pfrags apply for a transfer to a neighboring particle by queuing a transfer request. When the neighboring particle can accommodate the transfer, the OS informs the pfrag with a call to *Transfer-Granted*. In response to this call, the pfrag copies itself over to the Launch Pad (outgoing FIFO). If the fragment's intent was to transfer to the neighboring particle, the pfrag will then *DeInstall* itself from the current particle. Alternatively, the pfrag could choose not to *DeInstall*, with the end effect that the fragment propagates a copy of itself.

Transfer-Refused In the case where the neighboring particle can not accommodate the transfer, the OS at the current particle informs the fragment with a *Transfer-Refused* message. The fragment must then de-queue the request[14].

Update The basic vehicle for process scheduling on a *paintable* is the *Update* handle. Once a pfrag has been installed, the particle's OS will intermittently call the *Update* routine to allow the pfrag to adapt to changes in its environment. Ideally, all pfrags in a given particle would appear to be simultaneously interacting with each other while reacting to changes in the envi-

---

14. In the current spec, transfer requests do not time out. It is up to pfrag to de-queue the transfer request after receiving an answer from the OS.

ronment. In practice, the particles use time sharing among the pfrags to emulate this group dynamic.

On the periodic calls to *Update*, the pfrag senses the state of the environment via reads from the I/O space, compares that to its internal state and selects one of a set of predefined behaviors. Possible responses include adding or removing posts to the HomePage, transferring to a neighbor, deleting itself[15], spawning a copy of itself, or just doing nothing.

If *Update* is called with no parameters, it runs to completion before returning control to the OS. If a single integer parameters is supplied, it indicates the length of the time slice allotted to it by the OS, measured in clock ticks. In contrast to traditional time-share systems, this value is only a suggestion. The *Update* always runs to completion. Yet, it can be helpful for the pfrag to know that it has very limited time and to adapt its processing accordingly.

TABLE 3-1. OS Commands that all Process Fragments must support

Commands	Description

Install (*addr* OS)
 	
Install (*addr* OS,
        addr scratch)
	Install the process fragment
	    OS → trap address for OS supported functions
	    scratch → one time scratch memory bound to
	            the pfrag

DeInstall () DeInstall (*int* Time)	DeInstall a process fragment **Time** → period of notice measured in clock ticks
TransferGranted (*addr* FIFO)	Requested transfer has been allowed **FIFO** → address of output buffer for transfer
TransferRefused ()	Requested transfer has been refused

Update ()
 	
Update (*int* Time)
	Execute a round of adaptation.
	Read the state of the environment from the I/O space
	and select an internalized behavior in response.
	    Time → maximum allowable time before
	            returning (measured in clock ticks)

---

15. Process fragments can call their own DeInstall () functions.

OS

Control of each particle's resources is vested in a real-time operating system (OS) embedded into every particle. As with most single-processor architectures, the *paintable* OS regulates the flow of information, services event driven interrupts, schedules the processing of local executables and allocates shared resources.

The operation of the paintable OS can be grouped into four categories: general housekeeping, the interface to the network subsystem, the strategy for calling the pfrag handles, and the pfrag toolkit. The general housekeeping tasks include functions like intermittently de-fragmenting the RAM and the I/O space. The remaining three function groups are discussed below. Note that this discussion is pfrag-centric in that it only presents the level of detail necessary to author a pfrag.

Network Interface. Particles contain three segments of shared memory that are used by both the pfrags and the network subsystem. These are the 2 pfrag transfer FIFO's[16] and the I/O space. Access to this memory is synchronized by the OS which uses separate control strategies to synchronize the pfrags and the network subsystem.

On the pfrag side, the OS employs a toolkit approach to maintain indirect synchronous control over the pfrags. Rather than accessing the memory directly, all pfrags must use OS-supplied toolkit functions for reads/writes to the shared memory, effectively inserting the OS as a proxy.

On the network subsystem side, the OS must be signalled whenever a new Pfrag has arrived in the Bassinet, when a transfer from the Launch Pad is complete, or when updates for the I/O space have arrived. Likewise, the OS must inform the network subsystem when a pfrag is queued in the Launch Pad, the Bassinet is cleared and/or new posts have appeared on the local HomePage.

While the details of the interaction will be implementation dependent, the OS must have the ability to block network access to shared memory in order to guarantee several crucial elements of the processing environment:

- Any change in the number of neighboring particles[17] must be reflected in the number of mirrors in the I/O space.

---

16. The Bassinet and the Launch Pad.
17. This includes I/O ports posing as particles.

- Yet, when a pfrag begins its *Update* cycle, the entire I/O space must remain stable for the duration of the cycle.

- Pfrag posts to the HomePages of all the neighboring particles must ultimately be mirrored in the I/O space.

- Pfrag posts to the local HomePage must likewise be broadcast to the neighbors.

Pfrag Handles. The OS influences the behavior of the pfrags through calls to the five required pfrag handles (table 3-1 above). Two criteria guide the scheduling of these calls; mobility and adaptation. The mobility criterion derives from the need to maintain an amount of free RAM space sufficient to support a constant flow of migrating pfrags. The adaptation criterion seeks to model an environment where multiple pfrags interact concurrently.

Maintaining a sufficient amount of free RAM is the "Grim Reaper" component of the OS design. The designer must define cost functions to decide how much RAM to keep open, and to decide which pfrags to *DeInstall* when free space becomes too scarce[18]. Both these cost functions can vary from simple to complex. And both take advantage of the statistics that the OS can gather as it arbitrates transfer requests, shepherds the pfrags through the FIFO's and proxies the pfrags' access to shared memory.

The OS strategy for maintaining adaptation is traditional time sharing. The OS sequences through all the resident pfrags, dispensing a time slice with a call to the pfrag's *Update* command. *Update* commands run to completion and therefore define the temporal granularity of the time sharing. The OS can pass to the pfrag a suggested time limit and monitor compliance. But unless the pfrag triggers a trap condition[19], the pfrag designer can expect every *Update* call to complete without interruption. Be nice kiddies.

Pfrag Toolkit. The OS bundles a suite of services into the Pfrag Toolkit (table 3-2).The minimum set provides support for reading and writing the HomePage, reading the I/O space, requesting inter-particle transfer, and assorted maintenance functions (random number generation). Beyond this required minimum, the OS can

---

18. Pfrags frequently call their own *DeInstall* function in the course of their normal operation. And in many applications involving a closed set of pfrags, the "Grim Reaper" component of the OS is obviated.

19. The OS must anticipate a number of coding pathologies (eg. infinite loops) monitor against them, and signal a trap condition when one is found.

offer a suite of commonly used library routines (eg. a math package), with the caveat that the pfrags must respond predictably in particles where the requested function is not available.

TABLE 3-2. Minimum suite of Pfrag Toolkit functions

Commands:	Pfrag Toolkit
HomePage: Write	
boolean postEntry (pfragId, length, post)	Post a single entry on the HomePage. Returns boolean flag for success/failure. **pfragId** → integer ID number assigned to Pfrag **length** → number of bytes in post. **post** → array containing post (length bytes)
boolean repostEntry (pfragId, lenOP, oldPost, lenNP, newPost)	Alter an existing post Returns boolean flag for success/failure. **pfragId** → integer ID number assigned to Pfrag **lenOP** → number of bytes in current post. **oldPost** → array containing current post **lenNP** → number of bytes in updated post. **newPost** → array containing update post
boolean removeEntry (pfragId, length, post)	Remove a particular entry from the HomePage. Returns boolean flag for success/failure. **pfragId** → integer ID number assigned to Pfrag **length** → number of bytes in post. **post** → array containing post (length bytes)
int removeAllEntries (pfragId)	Remove all the posts linked to the given pfrag, Returns the number of posts removed. **pfragId** → integer ID number assigned to Pfrag
HomePage: Read	
int getHpSize ()	Return the number of posts in the local HomePage
int getEntryLen (entryId)	Return the number of bytes in a given post **entryId** → index to the selected post
boolean getEntry (entryId, post)	Read a post from the HomePage. Returns boolean success flag **entryId** → index to the selected post **post** → array to hold post contents
I/O space: Read	
int getIoSize ()	Return the number neighboring HomePages mirrored in the I/O space

TABLE 3-2. **Minimum suite of Pfrag Toolkit functions**

Commands:	Pfrag Toolkit
int getIoHpSize (hpId)	Return the number of posts in the selected mirrored HomePage **hpId** → index to the selected HomePage in the I/O space
int getHpEntryLen (hpId, entryId)	Return the number of bytes in a given post **hpId** → index to the selected HomePage in the I/O space **entryId** → index to the selected post
boolean getHpEntry (hpId, entryId, post)	Read a post from the HomePage. Returns boolean success flag **hpId** → index to the selected HomePage in the I/O space **entryId** → index to the selected post **post** → array to hold post contents
Process Fragment Transfer	
boolean queueTransfer (pfradId, hpId, pfragSize)	Request a transfer to the neighboring particle corresponding to the selected HomePage from the I/O space. **pfragId** → Pfrag ID originally assigned by the OS **hpId** → index to the selected HomePage in the I/O space **pfragSize** → total size of Pfrag (in bytes)
boolean queueTransfer (pfradId, hpId, pfragSize, scratchSize)	Request a transfer to the neighboring particle corresponding to the selected HomePage from the I/O space. **pfragId** → Pfrag ID originally assigned by the OS **hpId** → index to the selected HomePage in the I/O space **pfragSize** → total size of Pfrag (in bytes) **scratchSize** → additional scratch space required by Pfrag
boolean queueTransfer (pfradId, hpId, pfragSize, type, prio)	Request a transfer to the neighboring particle corresponding to the selected HomePage from the I/O space. **pfragId** → Pfrag ID originally assigned by the OS **hpId** → index to the selected HomePage in the I/O space **pfragSize** → total size of Pfrag (in bytes) **type** → ID for Pfrag type (see glossary) **prio** → transfer priority relative to Pfrags of same type
boolean queueTransfer (pfradId, hpId, pfragSize, scratchSize, type, prio)	Request a transfer to the neighboring particle corresponding to the selected HomePage from the I/O space. **pfragId** → Pfrag ID originally assigned by the OS **hpId** → index to the selected HomePage in the I/O space **pfragSize** → total size of Pfrag (in bytes) **scratchSize** → additional scratch space required by Pfrag **type** → ID for Pfrag type (see glossary) **prio** → transfer priority relative to Pfrags of same type

TABLE 3-2. Minimum suite of Pfrag Toolkit functions

Commands:	Pfrag Toolkit
void clearTransferQueue (pfragId, hpId)	Clear the transfer request. **pfragId** → Pfrag ID originally assigned by the OS **hpId** → index to the selected HomePage in the I/O space
Maintenance	
boolean copyMem (pfragId, fromAddr, toAddr, len)	Copy a continuous segment of memory **pfragId** → Pfrag ID originally assigned by the OS **fromAddr** → source address **toAddr** → destination address **len** → number of bytes
int getTime ()	Returns the system time (in clock ticks).
boolean markForGC (pfragId)	Mark the Pfrag for deletion by the OS. **pfragId** → Pfrag ID originally assigned by the OS
double getRandomDouble ()	Return a sample of a uniformly distributed random variable in the range [0.0 1.0]

External I/O (revisited)

All data exchanged between particles consist of pfrags in transit, updates for posts on the I/O space, or OS ↔ OS arbitration for pfrag transfer. I/O devices gain access to the particle ensemble by masquerading as particles, and must therefore dress there external I/O as either migrating pfrags or HomePage posts.

I/O portals necessarily script their wireless interactions to maintain the illusion that they have a HomePage, and protocol support for pfrag transfer. While indistinguishable at the signalling level, portals can still differentiate themselves from particles by posting "portal ID" posts to their pseudo-HomePage[20]. Pfrags in the vicinity sense these posts and interpret them as either a general announcement that the 'particle' is actually a portal, or as a message directed to selected pfrag types.

Input portals pass data either by posting to their pseudo HomePages or by streaming pfrags. Portals stream pfrags by flooding the neighborhood with transfer

20. A tag specifically for this purpose is defined in the simulator's development environment.

45

requests, following up every *Transfer-Granted* with a pfrag. Transfer requests from the neighboring particles to the portal are acknowledged but immediately refused.

Output portals receive data by either reading posts from neighboring particles, or by inducing migrating pfrags to transfer in. Pfrags can be attracted either by directed messages posted on the portal's pseudo HomePage, or by Gradient fields[21] radiating outward from the portal. Information in the HomePage posts of the Gradient pfrags can trigger selected pfrags to walk the gradient field back to the source — in this case, the portal.

Chapters 4 and 5 show examples of the portals performing the following functions:

- streaming of packetized data embedded as payload in pfrags
- sequencing the insertion of pfrags to direct the formation of structures.
- parallel image capture via an array of sensors scattered among the particles.

Run Time Example

Summarizing, *process fragments wander among the particles, sensing the environment through the visor of the I/O space. and reacting in accordance with a set of internalized strategies for interaction and migration. Interaction is restricted to posts on the HomePage. Migration is via requests for transfer to neighboring particles. Migration becomes propagation when pfrags transfer copies of themselves.*

This section illustrates the runtime behavior with an example of six pfrags interacting to execute a parallel data flow algorithm. The processing substrate is a series of particles regularly spaced along a line (fig. 3-8*a*). At one end is a single I/O portal, while the opposite end extends to infinity (read: is not relevant). The communication radius is sufficient for each particle to communicate with its two nearest neighbors.

In this illustration, the entry of the six pfrags is sequenced in such a way that they assemble into a processing structure which supports the 3-way concurrent processing of fig. 3-8*b*. Input vector data is streamed from a common source, analyzed concurrently by three processes, with the scalar results passed up to a fourth process for comparison[22]. This structure emerges from the interaction of three groups

---

21. Gradient fields are an elemental instance of pfrags that replicate and self organize into structures. They are described in detail in Chapter 4.

(a)

FIGURE 3-8. Architecture and Algorithm

(a) Processing substrate consisting of particles arranged in a line. At one end is an I/O portal with capable of mimicking a particle. The communication radius supports communication between adjacent particles.

(b) Data flow in concurrent process. Vector data is streamed from a common source to three concurrent processes. Each produces a scalar value as a result and passes that value along to an arbiter

22. This architecture is well suited to general problems of classification and modeling, where a stream of sampled data is concurrently compared against multiple models. For each model, a scalar error metric is accumulated and passed along to an arbiter process for comparison.

of pfrags: BreadCrumb, NearSightedMailman, and the KnittingClub[23]. Bread-Crumb builds a scaffold. NearSightedMailman delivers the input data to the several members of the KnittingClub, which perform the analysis.

BreadCrumb. BreadCrumb assigns to each particle a unique BreadCrumb-relative address. It operates by first depositing a copy of itself into every particle and then negotiating with its neighbors to arrive at a unique ID. This ID is posted to the HomePage and updated dynamically as needed. The goal is a sequence of addresses which is monotonically ascending.

This global behavior is easily implemented as a series of strictly local operations. On entry into a particle, BreadCrumb runs its *Install* handle. *Install* posts to the HomePage a post consisting of two words: a tag common to all BreadCrumb process fragments, and a crumb number (CrumbNr).
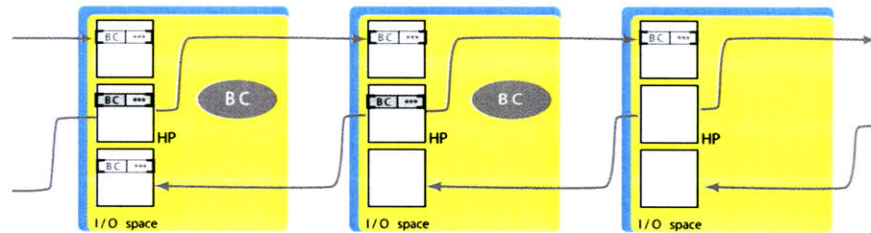
FIGURE 3-9. BreadCrumb pfrag: dynamic behavior

BreadCrumb pfrag spreads virally, installing a single copy of itself on every particle. It then interacts with the neighbors to build an ascending series of addresses. Note how posts to the local HomePage are mirrored in the I/O space of the neighbors

On subsequent calls to *Update*, BreadCrumb scans the I/O space and uses its contents to select one of three behaviors: propagation, adaptation, or removal (fig. 3-9). BreadCrumb selects the propagation mode if the I/O space indicates that one of the neighbors does not yet contain a BreadCrumb. Here, the pfrag requests a transfer to

---

23. A perk associated with doing novel work is the liberty to apply a warped sense of humor to the naming conventions.

the empty neighbor and returns. Until the transfer request is answered with either a *Transfer-Granted* or *Transfer-Refused*, ensuing calls to *Update* will simply confirm that the transfer is still pending and return. If the transfer is granted, BreadCrumb creates a copy of itself and queues that copy for transfer. Before releasing its copy, the originating BreadCrumb sets the CrumbNr. of the duplicate to be one greater than its own.

When both the neighboring particles contain BreadCrumb's, the pfrag examines the CrumbNr's from the neighboring posts and selects the smallest one. If this neighborhood minimum is greater than or equal to its own CrumbNr, BreadCrumb interprets this as a termination condition and removes itself by calling its own *DeInstall* handle. Otherwise, BreadCrumb adjusts its own CrumbNr to be one greater than the neighboring minimum, altering the local HomePage post as necessary.

The end effect of this strategy is that BreadCrumb erects a scaffold, with the CrumbNr's indicating the position.

Even a process fragment as rudimentary as BreadCrumb illustrates several characteristics essential to the programming model.

- **Adaptation**: Adaptation is a form of information transmission. The information processing tasks best suited for a *paintable* are those which map well to asynchronous agencies interacting locally to adapt to their environment. In this context, efficient adaptation is best served by short *Update* handles which execute smaller increments of adaptation but more frequently.

- **Exit strategy**: An important component of adaptation is the ability to void the particle ensemble of computations that are no longer relevant. *Paintable* pfrags often do this by incorporating a dependency chain capable of triggering a mass extinction if broken. In BreadCrumb's case, the dependency is the requirement that every pfrag always have at least one other pfrag in sight with an address smaller than its own[24].

- **Active data**: All posts to a HomePage are associated with a process fragment that is currently installed in the particle. Conversely, process fragments can not leave behind HomePage posts after they are *DeInstall*'ed.

  As instances of tagged data, posts have the expressive richness typically associated with data directed processing. However in the *paintable* architecture, the

---

24. The BreadCrumb with CrumbNr 0 is presumably nothing more than a post at the originating I/O portal.

intent is that the tagged data also exhibit reactive behavior — that they alter their content and position in response to a changing environment. Binding each post to the originating pfrag is the vehicle by which the posts are given a procedurally defined behavior.[25]

NearSightedMailman. The NearSightedMailman pfrag (NSM) is the prototypical message carrier. Prior to entry into the particle ensemble, it receives a packet of data as payload and a target address for delivery. Once in the ensemble, it proceeds to the destination, delivers the data packet in accordance with local instructions, and then deletes itself.

This function is realized as a pfrag with three behaviors; transport, delivery, and termination. Periodic calls to *Update* always begin with a scan of he I/O space. If no BreadCrumb posts are found, NSM interprets this as a termination condition and the pfrag calls its own *DeInstall*.Otherwise, if the CrumbNr. in the local HomePage does not match the destination address, NSM seeks a transfer to the neighbor with the closest match[26]. On arrival at the destination, NSM searches the I/O space, looking for posts containing coded instructions from client pfrags in the adjoining particles. It responds to these posts by either posting descriptive information about its payload, posting its payload, idling, ar *DeInstall*'ing itself.

All communication with the NSM pfrag is through HomePage posts. Any pfrag on one of three particles is a potential client. All the tags, formats and protocols must be known a priori by the clients.

KnittingClub. KnittingClub maps the parallel data flow of fig. 3-8*b* to the operations on the 1D particle ensemble. Input vector data is read in from a common source, analyzed concurrently by three processes, with the scalar results passed up to a fourth process for comparison. This organization naturally suggests an implementation as four pfrags. These pfrags must maneuver to the correct location, assemble into formation, solicit their input, and regulate their internal data flow. As necessary, they must also sense and respond to termination conditions.

The pfrags of KnittingClub are the "Hostess" and three "Members". Each of the Members implements one of the three analysis routines, passing the results along to

---

25. The best contemporary analogy would be consider the posts as windows to the *instance variables* of objects in an objected oriented language like C++ or Java.

26. The monotonic progression of the CrumbNr's insures that NSM will ultimately arrive at its destination.
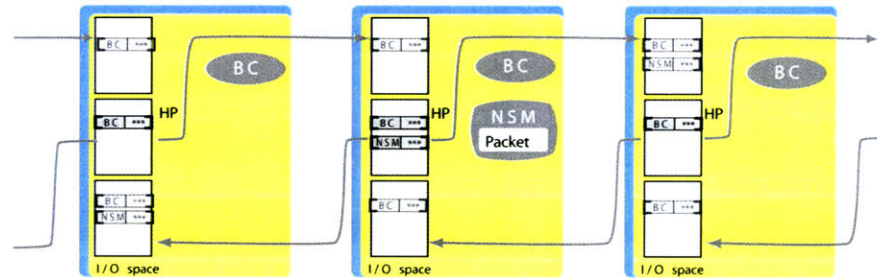
FIGURE 3-10. NearSighted Mailman

NearsightedMailman carries a data packet (embedded as payload) to a destination particle designated by a target CrumbNr. Once there, it scans the I/O space for posted instructions directing the disposition of its payload. In this figure, it has arrived at its destination and is idling while waiting for directions from client pfrags (which have yet to arrive).

the Hostess for comparison. The Hostess also regulates the flow of input data from the NSM and monitors the progress of each of the Members. Signalling among the pfrags is via posts to the HomePages. Pfrags within sight of each other[28] abuse the mutually visible HomePage as a broadcast packet network, using posts to announce their presence, report their state and receive their instructions.

Prior to entry into the particle ensemble, the Hostess must be told the ID of the NSM that it is looking for, and the codes for signalling with it. Once the Hostess and Members are passed into the particle chain[27], they collectively execute one of four operating modes: transport, formation, processing, and termination. In transport, all four pfrags orient themselves to the BreadCrumb trail. The Hostess walks the trail searching for the target NSM. The Members simply walk the BreadCrumb trail single file, stopping only when either the Hostess or another Member is in sight[28].

---

27. Hostess first

28. Pfrag **A** is "in sight" of a pfrag **B** if a post from the pfrag **A** is anywhere in the I/O space of the particle containing pfrag **B**.

51
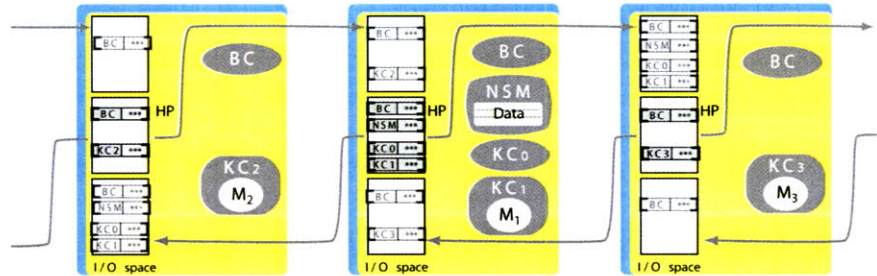
FIGURE 3-11. KnittingClub in formation

The Hostess (KC_0) takes up position in the same particle as the Near-SightedMailman. The Members (KC_{1-3}) spread themselves evenly over the neighborhood. Note that posts by NSM to its local HomePage are visible on all three particles.

Once the Hostess arrives at the particle containing the targeted NSM, it signals the formation mode. Members then position themselves such that no two Members share a particle, but that every Member can see the Hostess[29] (fig. 3-11). The processing mode commences when the Hostess signals the NSM for the first installment of the input data. NSM delivers the requested data via HomePage posts, removing one installment only after the Hostess has requested the next one. Members likewise employ HomePage posts to signal their status and pass their results. Processing continues in this vein until all the members have passed their end results to the Hostess.

---

29. An example of a Member rule set that produces this formation would be:
   1) walk up the address chain until you "see" the Hostess.
   2) record the CrumbNr. address of her particle (call it x)
   3) walk to the particle which has the greatest address in the range $[(x-1), (x+1)]$ and which is currently unoccupied.

Discussion & Related Work

This completes the definition of the programming model. And while many points remain open, it is not too early to revisit the criteria for success laid out in chapter 2. Two points are worth noting:

- The minimalist definition of a pfrag favors simplicity. Yet it is also complete. The OS toolkit commands are sufficient to construct pfrags that can maneuver among the particles, exchange data, procreate and die.
- The programming model is inherently scale agnostic. The processing environment within a given particle is not affected by changes in the particle topology outside its communication radius.

The HomePage mirroring technique of the *paintable* shares commonalities with established techniques for inter-process communication. Three representative examples are blackboards systems, tuple spaces, and reflective memory.

Blackboard Systems. Blackboard systems (BBS) are a technique for structuring the interaction of multiple agents popular within the AI community[15]. Defining elements are the blackboard, a controller, and a heterogeneous collection of experts. Input data, partial solutions, questions and notes, are aggregated into a single publicly viewable database — the blackboard. Experts incorporate domain knowledge and diverse strategies for attacking sub-parts of a problem. The controller gates access to the blackboard, enforces consistency and monitors the progress toward a solution.
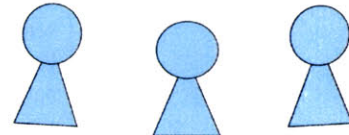
Scheduling is opportunistic. Experts (alone or in groups) petition the controller for access to the blackboard. Those granted access execute their processing, adding their results to the blackboard for use by succeeding experts. The overall solution strategy is incremental and naturally favors problem domains, such as image analysis, where the excessive combinatorics frustrate fixed procedural approaches[17].

It is natural to compare the blackboard and experts to the *paintable's* I/O space and pfrags. However the BBS experts are unbounded in size and complexity. Further, they enjoy a global view of the entire blackboard. Scheduling is likewise managed

with a controller with a global extent. Data deposited on the blackboard is static in that, once posted, it is independent of the fate of its originating expert.

This contrasts with the paintable PM where interaction between pfrags is restricted to simple posts, where the posts are temporally bound to the originating pfrag, where each interaction space is localized to the HomePages of a few particles, and where the pfrags are necessarily small and of bounded complexity.

Tuple Spaces. Tuple spaces were originally developed as the communications mechanism for a dedicated distributed processing language [19], and later integrated into a distributed extension of Java[36]. Messages are passed between independent processes operating asynchronously. A sending process tags a message with ID information and inserts it into a tuple space — a computational abstraction which functions as a publicly accessible repository. Messages are generative in that, once deposited into the tuple space, they exist independently of the sender. Both the message ID and format must be known a priori by the receiver. Receivers can request a specific message by name or use wild card descriptions to request a range of messages. Process membership in a tuple space is freely variable and each process can belong to multiple spaces.
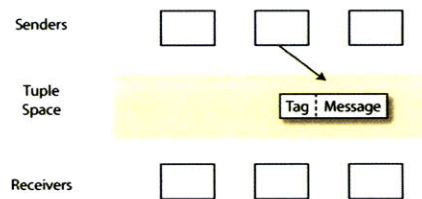
While multiple tuple spaces can emulate the networking of a *paintable*, the generative nature of the messages is a fundamental difference. Tuples can outlive the sending process, remaining in the space indefinitely until picked up by a receiver. By contrast, posts to the *paintable*'s HomePage are temporally bound to the originating pfrag. Further, when a receiving process extracts a message from the tuple space, it is no longer available to other receivers. Pfrag posts, on the other hand, have unrestricted availability for the duration of the post.

Reflective Memory Systems. In reflective memory systems, networked processing nodes communicate via writes to a segment of shared memory. Conceptually, the shared address space points to a single segment of physical memory. In practice, copies of the shared memory are maintained locally at every node, with dedicated hardware maintaining
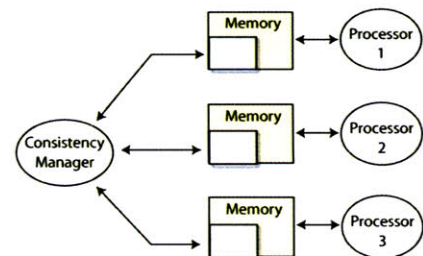
consistency[30]. Numerous variations on the basic theme derive from alternate approaches to consistency management[31]. Advantages are extreme fault tolerance, constant access times, and wide commercial availability. While the fundamentals inherently resist scaling, at least three implementations have been reported which promise network node counts into the thousands (Merlin[52], Sesame[53], and Shrimp[1])

Of the three techniques reviewed, reflected memory systems are the more distant cousin to the *paintable* PM. In reflected memory systems, locations in the shared address space are subject to writes from multiple sources, necessitating overhead to arbitrate collisions and insure synchrony in bounded time. By contrast, memory locations on the paintable HomePages are written to by one and only one processor node, and explicitly eschew bounds on the latency of the mirroring. It is self assembly from chaos, or nothing.

---

30. Reads from the virtual shared memory are free, writes trigger a round of 'reflection' to the remaining local copies.

Simulation Environment

Development and test of the process fragments was performed on Psim — a dedicated simulation environment written both for data gathering and as a pre-development tool for design of actual particle hardware.
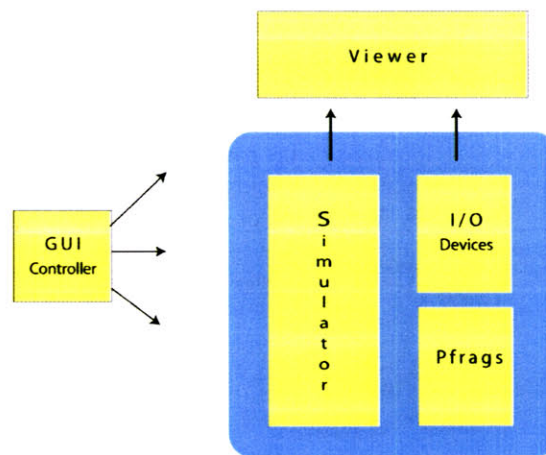
FIGURE 3-12. Psim Overview

Particles are modeled as Java objects organized in an ensemble of selectable size, density and dispersion. I/O devices run as a separate thread and serve as portals for the pfrags. A viewer assigns an icon for each particle and color codes the icon to reflect the contents of the particle's HomePage.

Psim is organized functionally into five Java packages; the simulator, I/O ports, pfrags, viewer and GUI/controller.

Simulator Particles are modeled functionally by instances of a template Java object. Particle objects are arranged in a 2D ensemble where the degrees of freedom include the number of particles, the communication radius and the placement variance relative to a regular lattice. The ensemble runs as a single thread which randomizes the order and duty cycle of time sharing among the particles[31].

I/O Ports All pfrags running on the simulator must enter the ensemble through an I/O portal. These portals are extensions of the Particle object, with added support for sequenced insertion of pfrags. At any point in a simulation, an I/O portal can be positioned graphically and integrate itself into the network of nearby particles.

Three types of portals are available; ports, devices and arrays. Single I/O ports can stream pfrags to/from a file. I/O devices consists of multiple ports whose behavior is coordinated by a shared state machine. Port arrays are larger ensembles of ports, with a density comparable to the particles and a single shared controller. Port arrays are well suited for parallel input to the ensemble (eg. image sampling via array of photo sensors).

Viewer Particles and I/O portals are assigned an icon by the viewer — a passive visualizer with access to each particle's HomePage and RAM space. The viewer periodically reads the HomePages and color codes the particle's icon under control of a programmable display function. An optional movie recorder can stream snapshots of the display into a file.

GUI/Controller Simulator functionality is assembled into a set of menu options. Menu items provide interactive control of execution (Run/Step/Stop), icon color coding, query of particle's state, movie authoring, and placement of the I/O portals. Subsets of the particles can be selected graphically with the mouse. Particles outside the selected region are deactivated by turning off their clocks. Additional selection/deselection can occur at any point in a simulation (fig. 3-13).

Pfrags Program files for the process fragments are grouped a single Java package. Each file defines a pfrag 'type' — a pfrag with the instantaneous state initialized to a common default value[32]. Pfrags are authored as Java objects with the required pfrag handles implemented as public methods. Every pfrag type is assigned a unique numeric ID. These ID's are cataloged in a single publicly readable file which is also stored in this package.

---

31. The simulator thread loops through calls to the ensemble-update, which in turn loops through the particles. The order in which the particles are called is randomized for every loop pass. Additionally, there is a nonzero probability that a particle will skip an update on a given loop pass.

32. Pfrags consist of the coded instructions which define their behavior, and their instantaneous state stored internally in a state vector. Two pfrags are of the same type if they use the same procedures to define their behavior.
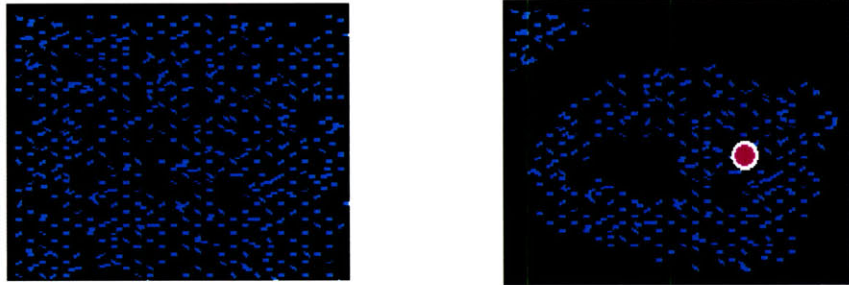
FIGURE 3-13. **2D particle ensemble:**
Initial distribution and subsets with I/O portal

Left: initial distribution of particles over a 2D rectangle. Viewer assigns icon to each particle and color codes it to reflect HomePage contents (blue → empty) **Right**: two subsets of particles are selected graphically via freehand sketch with the mouse. An I/O portal is positioned in one subset (blue → selected)

Experiments in this report were run on Psim version 0.6 — an interim implementation of the programming model with a number of restrictions:

- The OS on the particles does not enforce a minimum of free RAM space. Pfrags must monitor the available RAM and, in the case of impending overcrowd, transfer out or *DeInstall* themselves. The OS continually monitors RAM usage, but responds to congestion only by signalling an error.

- No OS-specific garbage collection. Version 0.6 falls back on the garbage collector native to Java.

- The pfrag handle *Update* always runs to completion, as opposed to being interrupted by the OS at the end of an allotted time slice.

- Inter-particle transfer proceeds directly, bypassing the FIFO's. On completion of the transfer-request / transfer-granted arbitration, a transferring pfrag is moved directly from the source particle to the destination particle. The latency is randomized by the simulator.

These restrictions were brooked in the interest of flexibility and speed of execution on contemporary workstation-class hosts. The subset implementation is well suited for exploring self-assembly in a regime where the pfrags naturally limit their density to small fraction of the available particle RAM — thus obviating the need for "grim reaper" component of the OS's function.

Pfrags written for the Psim environment assume the availability of the functions in the Java Math package. This implies that any hardware realization of the particles is likely to either run a lightweight Java VM, or support a subset of the functions in the Java Math library.

Summary

The purpose of this dissertation is to explain how to usefully program a paintable computer. This chapter distilled insights from material self-assembly into a detailed definition of a programming environment, a reference description of the computing hardware, and simulator suitable as a development environment.

Table 3-3 itemizes the key points of this chapter, organized by section.

TABLE 3-3. Key Elements in the System Architecture for a Paintable

Hardware Reference Platform	• Standard fetch-decode-write execution core operating on local memory. typically a low power microprocessor operating on memory with an address space of 50K-200K words.
	• Data exchange to neighboring particles via write/reads to virtual portals maintained by networking subsystem. • Number of portals is automatically adjusted to account for changes in number of neighboring particles within a communication range.
	• Inter-particle messages are error-free, but exhibit probabilistic transit times.
	• Power and communication subsystems support at least partial latitude in placement of the particles. No precision placement or interconnects.
	• Pushpin configuration as an exemplar of this architectural class. • 1 cm^3 size scale with pins contacting planes of layered composite. • Low technology threshold.
Programming Model	• Model based on three cornerstones: a particle internal OS, and two abstractions for program organization and shared memory.
	• All executables organized as process fragments (pfrags); mobile, autonomous program components. • Internalized procedures can be arbitrarily complex, but exposed interface is limited to 5 required methods and optional posts — messages encoded as key/value pairs.
	• Inter-process communication via posts to HomePage — reserved areas of local memory whose contents are eventually at mirror sites on the neighboring particles. • Posts from neighboring particles are assembled into the I/O space — a read only bulletin board-style area of local memory.

TABLE 3-3. Key Elements in the System Architecture for a Paintable

Programming Model (continued)	• Each particle has embedded OS that performs resource management and process synchronization. • OS coordinates access to memory pages and FIFO buffers that are shared with the networking subsystem. • OS proxies pfrag access to particle resources through a suite of "Toolkit" commands • OS uses method calls to pfrags to emulate time-sharing among pfrags.
Simulator	• Simulation environment written for code development and data gathering, and as a pre-design step for actual hardware. • Written in Java 1.1.7 and grouped into five packages: particle simulator, viewer, GUI/controller, external I/O, and pfrag library.
	• Simulator models each particle by separate instance of template Java object. • Objects arranged in 2D ensemble of selectable size, density, dispersion and communication radius.
	• All pfrags enter the ensemble via I/O ports — particle objects extended to support the insertion/removal of pfrags. Ports come in three flavors: • I/O portals are single ports for streaming pfrags between the ensemble and file storage. Useful for sequenced (but unsynchronized) insertion of pfrags. • I/O devices are portals that can be grouped and controlled centrally by a finite state machine. Useful for pfrag insertion coordinated over multiple ports. • I/O arrays are dense arrays of I/O ports, also centrally controlled by finite state machine. useful for dense sampling of physical stimulus (i.e. an array of photo sensors for image capture).
	• Visualization supported by a viewer package. • Each particle is assigned an icon which is color coded to reflect the contents of the HomePage. • Optional movie generation by streaming snapshots of the viewer to a file.
	• GUI control over execution (Run-Stop-Step), placement of I/O ports, and color map for viewer icons. • Subsets of the ensemble can be selected graphically and enabled/disabled. • Graphical selection of single particles for query of HomePage contents.

Essential Process Fragments

Software with the principal role of interacting with the physical world must, of necessity, acquire some properties of the physical world

   - Edward A Lee, U.C. Berkeley.

This chapter illustrates the design and operation of process fragments with six examples. In three examples, single pfrags propagate and coordinate to estimate distance and to diffuse packet data uniformly throughout a particle ensemble. In the remaining three examples, small sets of pfrags interact to tessellate a surface, build a communication channel, and construct a 2D coordinate system. In each of these examples, we define the task, lay out the implementation, and answer four questions: Does it work? Does it scale? What are its limits? Why is it important?

Along the way, we see examples of pfrags organizing into stable structures, directing data exchange within the structure, and doing so with a useful degree of scale invariance. Two basic programming abstractions are introduced: the *vfrag* and an *operator*. Vfrags are single pfrags that support virtual emulation of multiple pfrags. An operator is a set of interacting pfrags that has been grouped into a function module with a well defined interface to the external world. The pfrags presented here form the building blocks for the applications in the next chapter.

A final note to the nomenclature; process fragments are often named after the physical phenomena that they emulate. For clarity, process fragments are capitalized, while the corresponding physical phenomena is lower case. For example: *The <u>Gradient</u> process fragment radiates a <u>gradient</u> field outward from an anchor point...*

Gradient

The Gradient pfrag posts to the HomePage of every particle an estimate of the distance **d** between the particle and a fixed point of reference[1]. These reference points are necessarily few in number and are typically the I/O portals of external devices. The estimate should be both robust to unit failure in the particles and adaptive to broad changes in ensemble's topology. The operation should also be reversible in the sense that, when no longer needed, the Gradients release the particle's resources and are purged from the ensemble
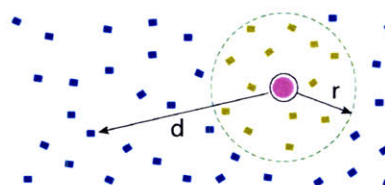
The Gradient enters the particle ensemble at the I/O portal as a single pfrag and propagates virally, ultimately installing a copy of itself in every particle in the ensemble. At any given point in the ensemble, copies of this process fragment communicate locally, building an estimate of the distance from the portal as a byproduct of their interaction.

The life cycle of a Gradient process fragment proceeds in four distinct stages; installation, propagation, adaptation, and removal.

Installation. In the installation phase, a gradient enters a particle and scans the local HomePage. If a post from another Gradient is found, the newcomer immediately marks itself for removal[2]. Otherwise, the Gradient posts to the HomePage a message consisting of five numbers (table 4-1).

---

1. This distance is normalized to communication radius
2. Asynchronies in the inter-particle networking can result in a situation were a Gradient process fragment enters a particle that was uninfected at the time the transfer was requested, but was infected during the interim.

TABLE 4-1. Generic post from Gradient

Post Name					
Gradient	Pfrag ID	ID #1	ID #2	HC (integer)	distance (real)

PfragID tag signifying that the Pfrag is of type Gradient

ID #1, #2 ID values assigned to initial Gradient and inherited by every copy.

HC hop count = distance from source portal measured in units of "communication radii" and rounded to the nearest integer.

distance distance estimate (real value average of all the proximal hop counts).

The PfragID is an identifier common to all Gradient pfrags. The two ID values are fixed by the originating portal and inherited unchanged among the propagating Gradients. In HomePages containing posts from overlapping fields, the ID's are used to associate a distance with a given source[3]. The hop count and distance estimate are constantly re-evaluated to automatically reflect local changes.

Once installed, the Gradient receives intermittent calls to its *Update* handle. It begins every call to *Update* by scanning the neighboring HomePages for posts from other Gradients with similar ID's. Based on the search results, it selects one of the three remaining phases.

Propagation. If the process fragment finds a particle whose HomePage is absent of a post from a Gradient, it applies for a transfer to the uninfected neighbor. If the transfer is granted, it creates a replica of itself with an incremented hop count and passes the replica onto the transfer, with the end effect that the Gradient blankets the ensemble (fig. 4-1).

---

3. External devices can interface to the ensemble through multiple portals. In this case, ID #1 is naturally a device specific ID, while ID #2 can specify a channel.
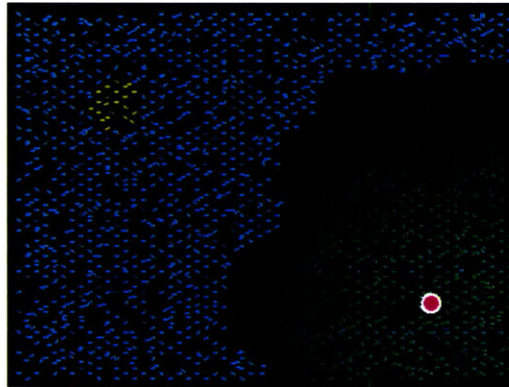
FIGURE 4-1. Gradient Propagation

A single copy of a Gradient enters through a portal and spreads through viral replication. Neighboring Gradients then interact to estimate distance from portal.

The group of highlighted particles in the upper left illustrate the number of particles in a single neighborhood. Particles with no pfrags are shown in blue. Green particles contain Gradient pfrags, with intensity of green inversely proportional to estimated distance to portal.

If there are no uninfected particles in view, the process fragment reads the hop counts from all the neighboring posts, selecting the smallest (HC_{min}). It then compares that value to the hop count value that it currently maintains in it post (HC_t) Based on the result of the comparison, a Gradient either adjusts the HC_t portion of its post, *DeInstalls* itself, or does nothing.

Adaptation. In adaptation, the values for the hop count and real distance are recomputed (formulas) and compared against the current values. The local post is updated as necessary to reflect any changes.

$$HC_{t+1} = HC_{min} + 1$$

$$HC_{total} = \sum_{i=1}^{N} HC_i + HC_{t+1}$$

Removal. If the scan of the neighboring posts fails to produce at least one post with an hop count smaller than its own, the Gradient treats this as a termination condition and calls its own *DeInstall* function.

$$D_{t+1} = \frac{HC_{total}}{N}$$

This continual re-evaluation affords the Gradient process fragment some ability to adapt to changes in the environment, such as a change in the shortest path back to the portal. Fig. 4-2 illustrates this adaptation. The simulator graphically selects an initial subset of the particle ensemble. In the steady state, a copy of the Gradient resides in each of the active particles and builds an estimate of the shortest path distance. When the remaining particles are reactivated, the Gradients automatically spread outward to the remaining particles, updating the distance estimates to reflect the new minimum length paths.

(a)   (b)   (c)
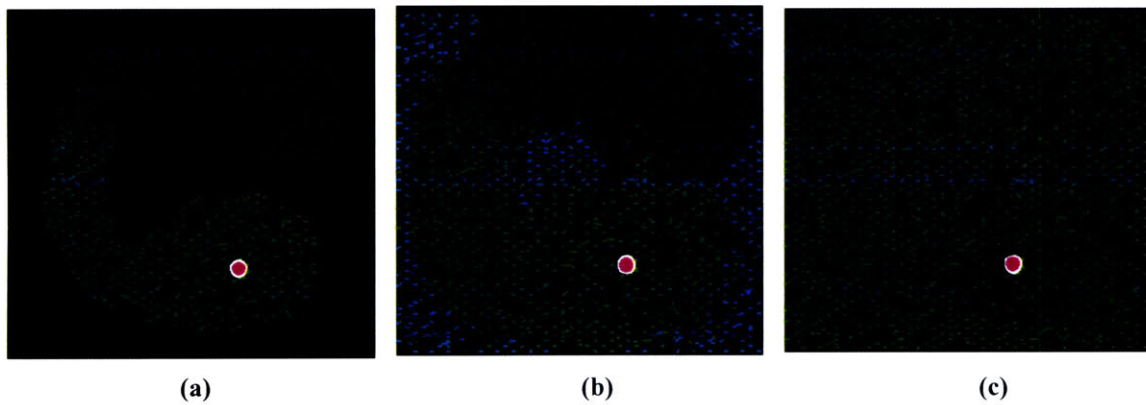
FIGURE 4-2. Gradient adaptation to change in topology

(a) steady state distance estimate on sub-region of particles.
(b) transient response to reactivation of remaining particles.
(c) new steady state

Note that the portal through which the original Gradient passed contains the only instance of a Gradient with a hop count of zero. Removal of the portal therefore triggers a mass extinction of the Gradients, whose dependencies all trace back to the post on the portal's pseudo HomePage (fig. 4-3).

The accuracy of the estimate is dependent on the density of the particles, and the anisotropy of the local communication region. The performance of gradient based distance estimates has been investigated by Nagpal [40], who showed that analytic results from work in packet radio networks could be employed to establish limits on accuracy of the estimates.
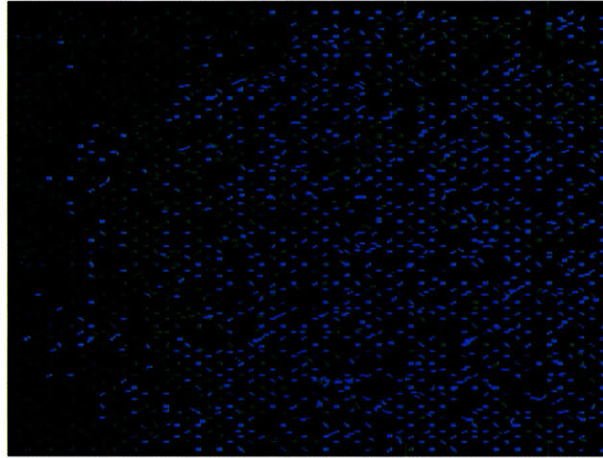
FIGURE 4-3. Gradient Removal

All Gradients ultimately rely on the ability to continually locate at least one neighbor with a hop count whose value is one less than its own. Failing this, the Gradient marks itself for deletion. This implied dependency chain traces back to the portal, which contained the only instance of a hop count of zero. The disappearance of the portal point triggers a mass extinction. Particles where the Gradient have *DeInstall'ed* are colored blue.

Distance estimates are of obvious import to any system relying on the relative placement of mobile components. In this work, we use distance estimates from the Gradient for both scaffolded and thermodynamic positioning. In both cases, the initial condition is a particle ensemble with multiple I/O portals at fixed locations, each emanating a Gradient (eg. fig.4-4). Pfrags seeking to position themselves relative to these fields, read the distance estimates from the I/O space and apply some internal decision function to select the next particle to transfer to. This process repeats until the pfrag has found the optimal match to its predefined rest position.

In scaffolded positioning, the wandering pfrag reads from the local posts both the distance and the absolute position for each portal[4]. Simple geometric constructs (eg. triangulation) are used to estimate an absolute position for each neighboring particle. The neighbor closest to the pfrag's target position becomes the destination for the next transfer. The runtime illustration from the last chapter (our good friends

---

4. This presupposes that the assigned absolute positions are consistent.
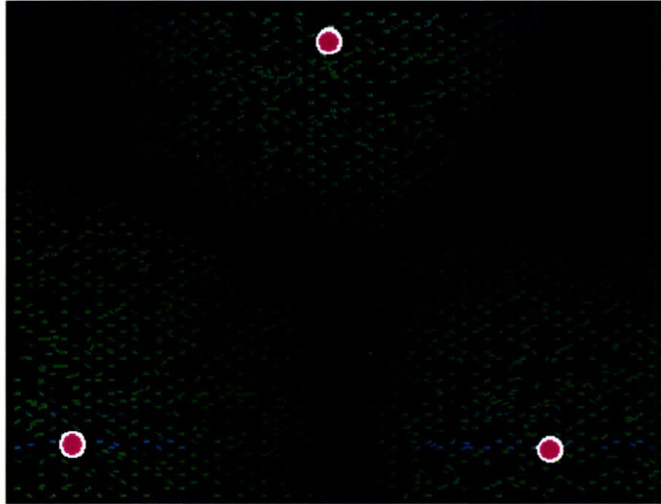
FIGURE 4-4. Multiple portals radiating overlapping Gradient fields

Three portals each radiate a Gradient field with a unique ID. In the steady state, every particle contains a Gradient for each portal. Intensity is inversely proportional to the distance to the closest portal.

BreadCrumb, NearSightedMailman and the KnittingClub) were toy examples of scaffolded positioning, with BreadCrumb building the scaffold.

In thermodynamic positioning, no absolute coordinates for the portals are required. The wandering pfrag uses the distance estimates to compute forces in a conservative field[5]. Two examples of these forces are the spring force, and the Coulomb's force for a point charge.

- Spring Force $\qquad$ $F = kd$

- Coulomb's Law $\qquad$ $F = \left(\dfrac{\pi \varepsilon}{4d^2} q_0 q_1 \right) \widehat{d}$

The pfrag then directs its migration towards the particle with the local minimum in free energy. As an example, consider the instance of a pfrag trying to position itself

_____

5. This approximation is only valid as long as the shortest uninterrupted path between the portal and the particle is a straight line.

equidistantly from the three anchor portals of fig 4-4. The pfrag could interpret each distance as a spring force and migrate toward that particle with the minimum potential energy

· Potential Energy
$$U = \frac{1}{2} \sum_{i=1}^{3} k_i d^2$$

where k_i = weighting function

Drawbacks. While Gradients adequately support a core functionality, there are some onerous practical limitations. When positioned in a particle, each Gradient consumes space on the HomePage (for posts) and in the particle's RAM (for the process fragment). For every portal that radiates a Gradient, space is consumed on every particle in the ensemble. This approach has a hard scaling limit and is only tractable for applications where a small number of external portals are actively emanating Gradients.

This is also impractical in the case where an arbitrary process fragment would emanate a gradient field at irregular intervals from a changing position. Any non-Gradient pfrag that would radiate a Gradient about its current position must essentially carry around a full copy of the Gradient pfrag as baggage. If such a process fragment were of the type that migrated frequently, the excess baggage would translate to excess demands on inter-particle communication bandwidth.

MultiGrad

MultiGrad is a single pfrag that mimics the behavior of an arbitrary number of Gradients. A single MultiGrad can be passed into the particle ensemble at any point. It then spreads virally, ultimately depositing a single copy of itself into every particle. Conceptually, this thin layer of MultiGrads can be regarded as an omnipresent ether. Other process fragments can radiate a gradient into this ether by simply posting a message to their local HomePage. MultiGrad treats this message as a point source and propagates a field. The degrees of freedom include the range of the gradient field and the contents of a message that the field radiates as payload

The implementation of MultiGrad starts with two observations:

1. Like all process fragments, Gradients communicate via posts to the local Home-Page.

2. All the state data necessary to distinguish one Gradient from another is contained in these HomePage posts.

MultiGrad exploits this by introducing the concept of a "vfrag" — a pfrag which uses time sharing to emulate the behavior of multiple pfrags[6]. In this case, the MultiGrad vfrag is emulating a variable number of Gradients. Each Gradient is represented by a vGradient, which can be regarded as a Gradient post without the associated Gradient pfrag[7]

Ideally, there would be no difference between a vGradient and the posts from a real Gradient. However two implementation details confound adherence to this ideal:

1. We need a mechanism to unambiguously signal to the vfrag. This is required for vfrag ↔ vfrag communication, and for pfrag ↔ vfrag communication.

2. The vfrag "has a life of its own" in that, independent of any pfrags that it might emulate, the vfrag must also be able to diffuse into the particle ensemble, propagate, operate, and ultimately be expunged.

---

6. Vfrag is short for "virtual pfrag". The definition of a vfrag as time shared emulation of multiple pfrags is inspired by the tradition of a "virtual machine" — a concept ubiquitous to the design of multi-user operating systems.

7. This may at first appear to violate a rule set forth in the previous chapter; namely that all posts are associated with a pfrag that endows the post with reactive behavior. In the case of a vGradient, the associated behavior is managed by the MultiGrad vfrag. If the vfrag is *DeInstall*'ed, all the associated posts go with it.

The actual operation of MultiGrad is reflected in its four posts, shown in table 4-2. Three of the these posts are written and managed by MultiGrad. The fourth post — the MultiGradCenterPost — is sourced by other pfrags as a means of signalling to the MultiGrad's.

TABLE 4-2. Four posts used by MultiGrad

Post Name						
MultiGradPresent	Pfrag ID					
MultiGradExiting	Pfrag ID					
MultiGradCenterPost	Pfrag ID	CD	Fs	sFlag	payload length	payload ...
MultiGradPost	Pfrag ID	CD	Fs	sFlag	payload length	payload ...

PfragID   tag signifying that the Pfrag is of type MultiGrad

CD   *count down* := field strength normalized in communication radii and rounded to the nearest integer. greatest at the source.

Fs   *field strength* := remaining amount of distance that the gradient field will propagate before terminating. (real number normalized to communication radius)

sFlag   *stability flag* := boolean variable indicating the settling of the *Fs* value. TRUE → *Fs* unchanged for the last 5 update cycles.

payload length   number of words in the attached payload

payload   variable length attachment to post

MultiGradPresent. On entry into an unoccupied particle, MultiGrad posts a MultiGradPresent message into the particle's HomePage. This single word post indicates to local process fragments that there is support for transmission and reception of gradient fields.

MultiGradExiting. Unlike the Gradient pfrag, a Multigrad does not have any implicit dependencies that can be employed to trigger a mass extinction. This is a necessary artifact of its support for the creation and removal of multiple vGradients. MultiGrad therefore employs an explicit messaging technique for mass removal from the particle ensemble.

From any particle in the ensemble, a MultiGradExiting message can be posted to the local HomePage[8]. Once a MultiGrad senses this post on a neighboring HomePage, it removes all the vGradients that it is servicing and puts up a MultiGradExiting post on its local HomePage. Once every neighboring HomePage contains this post, the MultiGrad marks itself for deletion.

MultiGradCenterPost. This is the vehicle by which pfrags initiate the propagation of a vGradient. Any pfrag can signal the creation of a vGradient by posting a MultiGradCenterPost on its local HomePage. The count down (CD) indicates the desired range of the vGradient in units of communication radii. The field strength (Fs) is initial strength of the field. (In a MultiGradCenterPost, Fs is just a real number version of CD). The variable length payload is propagated along with the vGradient as a means of uniquely disambiguating it from other vGradients.

Because the MultigradCenterPosts are written by other pfrags, MultiGrad treats this post as read-only. Removal of the MultiGradCenterPost automatically triggers the removal of the associated vGradient.

MultiGradPost. This post is both a vehicle for signalling between MultiGrads on neighboring particles, and a means by which other pfrags can detect and read a vGradient. MultiGrad propagates the vGradients in the MultiGradPosts. In a particle within the range of one or more fields, each field is represented by a single MultiGradPost. As the field propagates outward from the source, both the CD and the Fs values in the post are decremented.

Pfrags within sight of a MultiGradPost can read the post to answer two questions: "How far away am I?" and "From what?. The *Fs* value in the post indicates the distance of the current particle relative to the source of the field[9]. The payload can be read to receive a message from the pfrag that initiated the field. The payload data also serves as a unique ID for the field.

The accuracy of the distance estimates in the vGradients is, by definition, identical to that of the Gradient. However MultiGrad offers substantially improved scale invariance, albeit at a higher initial cost in particle resources. Table 4-3 compares the RAM usage[10] and the HomePage requirements[11] for the two pfrags. This table

---

8. In this work, the MultiGradExiting post is always introduced in a I/O portal.

9. This is not strictly true. In addition to the field strength (*Fs*) a querying pfrag must also know the original field strength at the source. This omission was a design error which is usually compensated for by including the original *Fs* in the payload.

separates the costs for a single gradient field from the incremental costs for additional fields.

The relative requirements favors the Gradient in cases where the number of overlapping fields is small. However, as the number of fields rises, the MultiGrad becomes more attractive and the limiting resource becomes space on the HomePages.

TABLE 4-3. Resource usage for Gradient and MultiGrad (in units of words)

	first field		additional fields	
	RAM	**HomePage**	**RAM**	**HomePage**
Gradient	450	5	450	5
MultiGrad	2200	20	0	9

The real import of MultiGrad is that it empowers any pfrag to radiate a gradient field over a selectable distance. This functionality plays a key role throughout the remainder of this report ... including the next section.

---

10. RAM usage for a pfrag is estimated by disassembling the Psim java code for the pfrag. This figure is a coarse estimate based on unoptimized code.

11. HomePage usage is estimated by counting the words in the posts of tables 4-1 and 4-2 and assuming a length of 5 words for the variable length payload of MultiGrad.

Tessellation Operator

The Tessellation operator groups the particles into the Voronoi regions about a uniformly distributed set of anchor points[12].The density of the anchors is selectable. The anchor distribution and particle region assignment automatically adapts to intermittent unit faults and global changes in the topology.

The *paintable* tessellation operator is realized as a combination of Centroids and MultiGrads. Centroid is a pfrag that searches for the particle which is equidistant from its immediately neighboring Centroids. In the absence of a MultiGrad, the Centroids must migrate blind, with no knowledge of the world outside their immediate neighborhood and no way to infer distance or orientation for the other Centroids that it can see.

The inclusion of MultiGrad affords each Centroid an opportunity to radiate a field containing ID information and an estimate of field strength that is linearly proportional to the distance. Centroids interpret the field strength from their neighbors as a repulsive spring force and seek the position which minimizes the potential energy

$$U = \frac{1}{2}\sum_i x_i^2$$

x_i is the field strength of Centroid

i in the set of neighboring Centroids in sufficient proximity for their fields to be read.

For any given particle, the region membership can be computed from the MultiGrad posts in its HomePage. The message payload of a MultiGrad post lists a unique ID value for originating Centroid. Pfrags in intervening particles can find the MultiGrad post with the strongest field strength and adopt the associated Centroid ID as a region label.

12.In traditional classification, a tessellation operator applies a few basic shapes to tile a continuous space into a regular pattern. We abuse the word "tessellation" to include Voronoi cells of varying shape that spaced about uniformly distributed centroids.

While this procedure performs adequately in the abstract, a number of subtleties must by observed to insure stability.

- *MultiGrad settling time* On Entry into a particle, a Centroid posts a MultiGrad-CenterPost which initiates a radiating field. The suitability of the new position is only apparent after the Centroid has an opportunity to affect its neighbors and sense their response. To this end, the Centroid idles for several update cycles before evaluating the neighborhood for a possible move.

  This is coarsely analogous to physical systems undergoing reversible change, where thermodynamic equilibrium must be approximately maintained during all stages of the change.

- *Enforced Randomness* The number of skipped update cycles is randomly perturbed about some fixed baseline. For the experiments shown here, ± 2 cycles was added to a baseline of 5. This breaks up any regularities that might otherwise cause cyclic behavior.

- *Viscosity* Large moves cause field fluctuations which can impede the settling into a local minimum, especially when the system is in the vicinity of that minima. Centroids avoid this hazard by favorably weighting moves over short distance, a technique that amounts to subjecting the moving particles to viscous drag.

  Information for determining the relative distance between particles is not natively supported by the particle's hardware but can be recovered from the MultiGrad posts.

As is the case with single pfrags, the dynamics of the dual-pfrag tessellation proceed in three distinct stages: installation, evaluation, transfer. Table 4-4 illustrates the activity of Centroid and MultiGrad in each of these stages.

TABLE 4-4. Tessellation: Interaction between Centroid and MultiGrad

	Centroid	MultiGrad
Installation	• Post MultiGradCenterPost • Skip *n* Update cycles where $n = W + \Delta$ $W = 5$ Δ = random number in the range [-2, 2]	• Field radiating • • • •
Evaluation	• Evaluate all neighbors (including current particle) • If best location is in neighboring particle, apply for transfer	• Field stable • • • •
Transfer	• Remove MultiGradCenterPost • Transfer to selected neighbor	• • Field decaying • •

The performance of the tessellation operator was qualified by tests run on the simulator. The 2D particle ensemble was defined consisting of 1030 particles. The communication neighborhood contained an average of 15 neighbors. As an initial condition, a contiguous subset of these particles was selected, with the clocks on the remaining particles turned off.

Into this sub-region, a set of 30 Centroids were diffused through a single input port. In the absence of the MultiGrads, these Centroids simply wandered about randomly, each seeking a position from which no other Centroids can be seen (fig. 4-5a).

A single MultiGrad was introduced at a second input port and spread throughout the ensemble. On sensing the presence of a MultiGrad, a Centroid would radiate field strength with a center value of 3, search for field data from other Centroids, and migrate towards the position which minimized the free energy. The Centroids ultimately arrived at the stable configuration shown in fig. 4-5b.

With this configuration as a starting point, the previously inactive particles were reactivated. The MultiGrad's are the first to react to this by virally spreading among the newly activated particles. The Centroids then follow (fig. 4-6a) ultimately arriving the new stable configuration (fig. 4-6b).

As the figures suggest, this simple dual-process fragment operator only weakly approximates the patterned regularity typically associated with tessellation. It is perhaps more analogous to a collection of spheres packed into a container. And while it is demonstrably adaptive, this rudimentary implementation of a tessellation operator has some limitations worth noting:

- Like all operations involving gradient fields, field data radiated via MultiGrad has a resolution which is statistically bounded by the number of particles in an average communication radius. This affects the positional accuracy, and by extension, the resolution of the estimate for potential energy.

- A pitfall unique to this implementation is the dependence on static values for initial field strength. Centroids radiate a field by posting a MultigradCenterPost as a signal to the neighboring MultiGrads. This post specifies an initial field strength which, together with the communication radius, determines the total distance that the field will radiate. In this bare-bones version of Centroid, that initial field strength is hard coded into the process fragment, resulting in a finite field range. This limitation on the field strength imposes both an upper and lower bound on the density[13] of the Centroids. Too many Centroids, and their associated fields exhaust the space on the HomePages (table 4-3). Too few Centroids, and their field strength is insufficient to blanket the entire ensemble (fig. 4-7). Nevertheless, for densities within these scaling bounds, the performance is independent of the total number of particles.

---

13. The ratio of number of Centroids to number of particles.
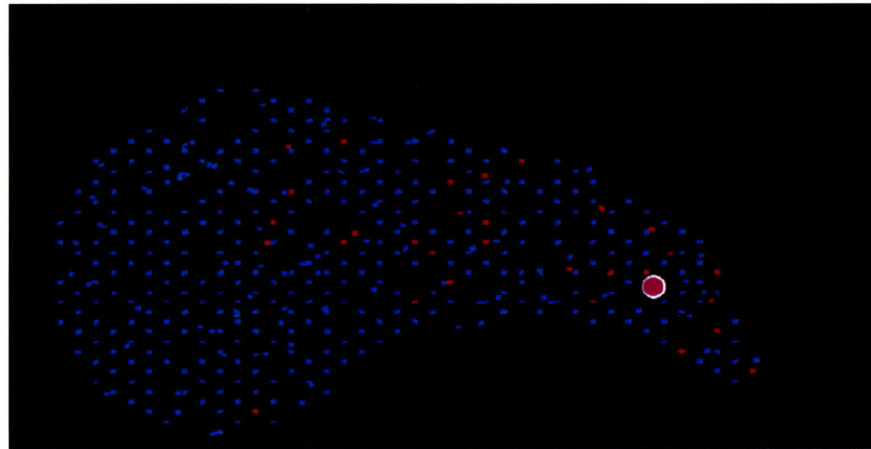
(a)

(b)
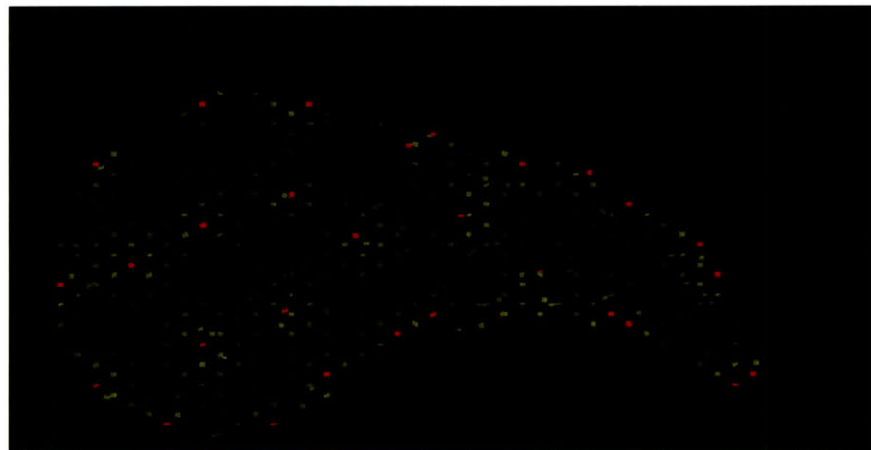
FIGURE 4-5. Tesselation operator applied to sub-region

(a) 30 Centroids (red) enter through a portal and randomly diffuse among a subset of the particles. (b) inclusion of MultiGrad enables tessellation. the intervening particles color coded according to field strength.
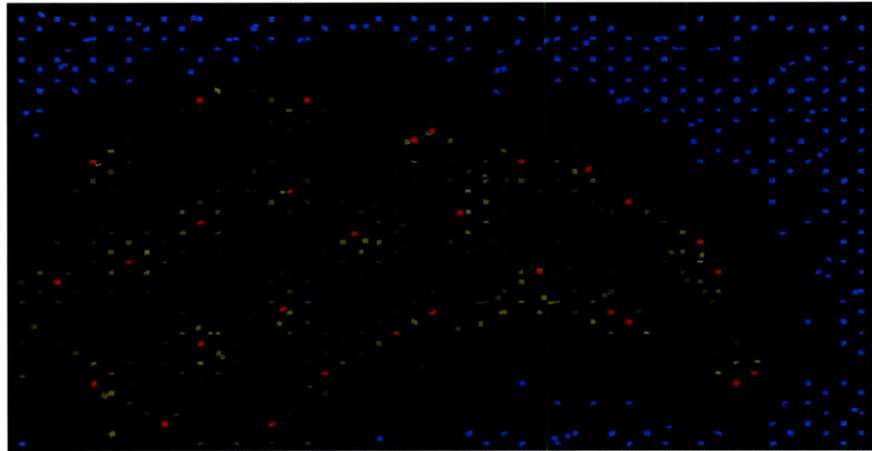
(a)
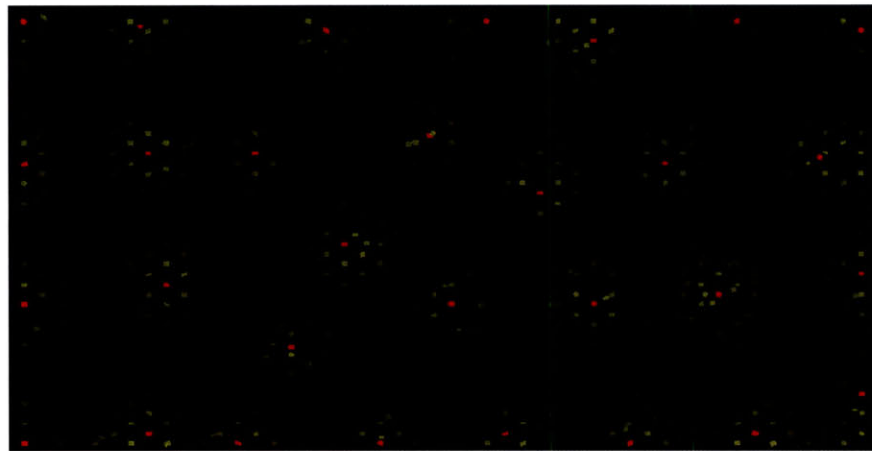
(b)

FIGURE 4-6. Tessellation: Adapting to expanded ensemble size

(a) Previously deactivated particles are reactivated (light blue). MultiGrads begin to propagate outwards (dark blue), followed by the Centroids (red).
(b) New steady state distribution

Several corrective options are available, all coming with a price. One option would be an extension to MultiGrad to support radiating fields with unbounded range. This would support true scale invariance, but at a price of overcrowded HomePages and an attendant limitation on the number of Centroids. A more attractive option is the inclusion of a third process fragment that estimates relevant global statistics of the ensemble which the Centroids, in turn, use to adaptively adjust the field range.
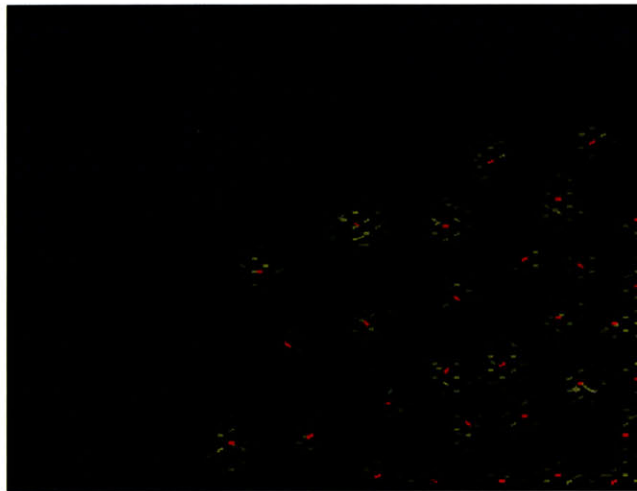
FIGURE 4-7. Tessellation: Insufficient field range

Fields radiating outward from the Centroids can have an insufficient range to collectively blanket the particle ensemble.

The import of the tessellation operator is its utility as a basic tool. It employs thermodynamic self organization to provide the scale resilient grouping operations that are fundamental to constructing complex structures. For example, in systems of heterogeneous particles, it is particularly well suited for tasks such as property assignment.

Diffusion

Diffusion is a single process fragment that functions as a carrier. It accepts a packet of data as a payload and employs a migration strategy for transporting the payload throughout the particle ensemble. The migration strategy is an ongoing shuffle driven by local disparities in the pfrag density. A group of Diffusions carrying a sequence of packets position the packets in such away that the packet density is uniform[14], and the position is constantly randomized.

The life cycle of a generic Diffusion is open ended. Typically, a serial stream of Diffusions enters the particle ensemble at a single point of entry[15]. Once in the particle ensemble, the pfrag hops from particle to particle. For every particle, Diffusion proceeds through three phases; installation, evaluation, and transfer.

Installation. On entry into a particle, a Diffusion places a single post on the Home-Page. This post announces the pfrag's presence and provides sufficient state data for multiple Diffusions to coordinate their activity. Table 4-5 lists the four words of the post. Three of these words, the PfragID, the StreamID and the Timer are symbols passed between Diffusions to coordinate their behavior. The fourth, Index, is for visualization only.

TABLE 4-5. Diffusion post

Post Name				
Diffusion	PfragID	Stream ID	Index	Timer

PfragID   the identifier common to all Diffusions

StreamID   identifier unique to a stream of packets

Index   identifier for individual packet within a stream

Timer   counter indicating the number of update cycles this pfrag has been in the current particle.

---

14. With a local variation of ± 1 packet.

15. For example, an I/O portal.

The Stream ID enables separate treatment of multiple streams of data (i.e. a given Diffusion pfrag will only interact with other Diffusions posting the same stream ID) The Index is a label for the payload The Index and the streamID combine to uniquely identify the payload. The Timer supplies an approximate indication of how long a process fragment has been in a particle. The unit of time is an *Update* cycle — the time required for the particle's OS to cycle through the *Update* handle of all of its process fragments[16].

Evaluation. Throughout its tenure in a particle, Diffusion's sole activity is to be looking for the next destination and to keep its place in line. On calls to its *Update* handle, Diffusion counts the number of other Diffusions in the current particle, and take a census of the local neighborhood. If no neighbor is has a lower occupancy than the current particle[17], or if there is a Diffusion in the current particle with a higher Timer count, the pfrag increments its own Timer value by one and returns. Otherwise, Diffusion requests a transfer to that neighbor with the lowest occupancy[18]. Fig. 4-8 illustrates this dynamic with the case of three Diffusions operating in a single particle.

The destination selection behavior differs somewhat for the process fragments located in an I/O portal. In the process of streaming a Diffusion into the particle ensemble, an I/O portal will create the diffusion in its RAM, and wake it up with a call to its *Update* handle. The only indication that the pfrag has that it is not in a particle is the presence of a IOPortal post in the local HomePage. In this case, it randomly searches the neighborhood applying for a transfer to the first particle it finds with an occupancy below a predefined threshold[19].

Transfer. After negotiating permission to transfer to a neighboring particle, the Diffusion removes its own post, and marks itself for removal to the transfer queue.

---

16. This unit of time is necessarily variable and is only suited for comparisons between Diffusions co-located in the same particle.

17. The occupancy count includes only those Diffusion process fragments with identical StreamIDs

18. Ties are broken by random selection, using a random number requested from the particle's OS.

19. This is but one of several possible approaches for streaming pfrags through an I/O portal. But in all cases, the pfrag's behavior as seen from the neighboring particles must be well defined.
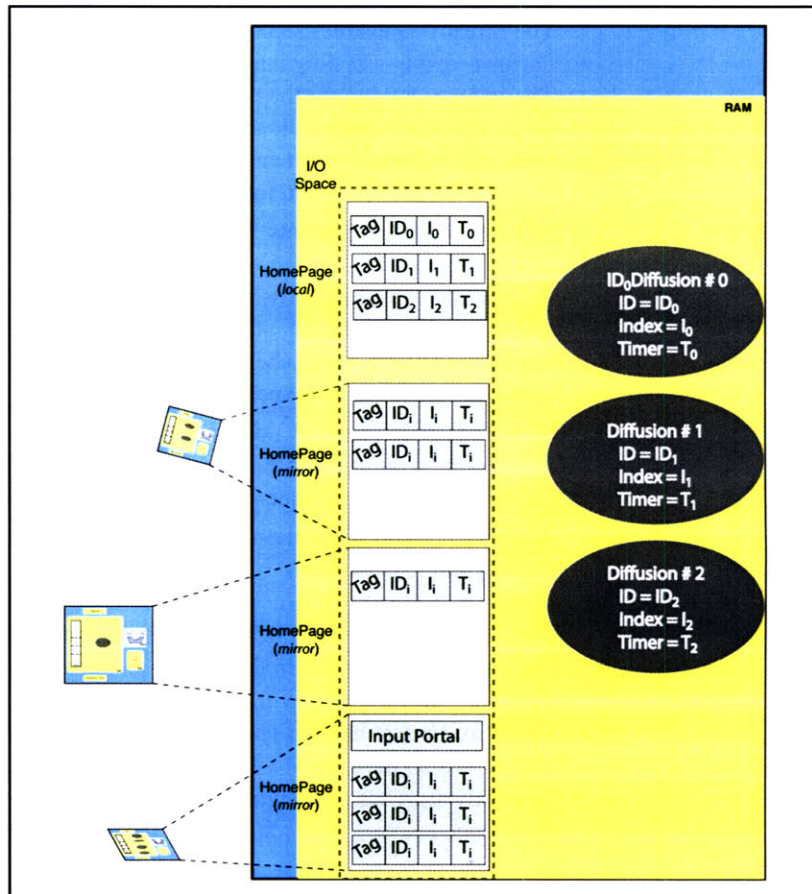
FIGURE 4-8. Diffusion: Rules for the Selection phase

Three Diffusions share RAM space in a single particle. On calls to their *Update* handles, each pfrag reads the Timer counts from posts on the local HomePage. The pfrag with the maximum Timer count searches the I/O space for the neighboring particle with the smallest number of Diffusion posts. If the minimum post count is smaller than the local post count, that Diffusion applies for a transfer. The remaining two Diffusions simply increment their Timer count and return.

Diffusions interpret an IOPortal post as a sign that the would-be particle is actually an I/O portal and disregard it as a possible transfer destination.

Diffusion pfrags as a group exhibit two behaviors; the transient behavior when the group is streaming into the particle ensemble, and the steady state behavior after the streaming is complete. Fig. 4-9 shows an instance where a sequence of Diffusions is streamed through a single portal. In this test configuration, each of the 660 particles is communicating with 10 to 16 of its immediate neighbors. Each particle is represented by an icon that is color coded to reflect the resident Diffusion with the greatest Index value[20]. A single I/O port passes into the particle ensemble a stream of 1500 Diffusions with Index's numbered linearly from 1 through 1500.

In the transient phase, data streaming in through the I/O port creates a density imbalance centered at the I/O port, putting a uniform outward pressure on all the pfrags. The result is that the distance of any given pfrag from the I/O port is proportional to the elapsed time since the process fragment passed through the port. The closer a pfrag is to the beginning of the stream, the farther it is from the port. In the transient stage, the distribution approximates concentric rings (fig. 4-9a) while the density is radially monotonic and centered at the portal (fig. 4-9b).

When all the data has passed through the I/O port, the transient phase yields to the steady state. The concentrated disparity in density vanishes and is replaced by pockets of localized residual disparity (fig. 4-9c and 4-9d). This residual disparity drives random motion among the process fragments. In the steady state, a small percentage of the process fragments are always moving and the position of any individual Diffusion is randomized in both space and time. This technique is comparable to a deck of cards undergoing a continual shuffling.

Diffusion suffers from the obvious upper bound on its scaling. Too many Diffusions fill the particle ensemble, at which point the I/O portal becomes congested and the shuffle responsible for randomizing the position breaks down. A more subtle pathology can occur when the number of pfrags equals an integer multiple of the number of particles. Recall that the ongoing shuffle is driven by local disparity in the density. If that disparity vanishes, the shuffling stops[21].

Diffusion is useful in applications involving heavy data transport. In its basic form, it is a very light weight carrier. In practice, this simple form is used as a template for the design of a more complex functionality. We will see this in the first application of the next chapter.

---

20. Comparable to ink staining the pfrags for in vivo observation.
21. Consider the case of 10 particles; 8 particles with 3 pfrag, 1 particle with 4 pfrags and the remaining particle with 2. Pfrags from 9 particles will apply for transfer into the 'hole'.
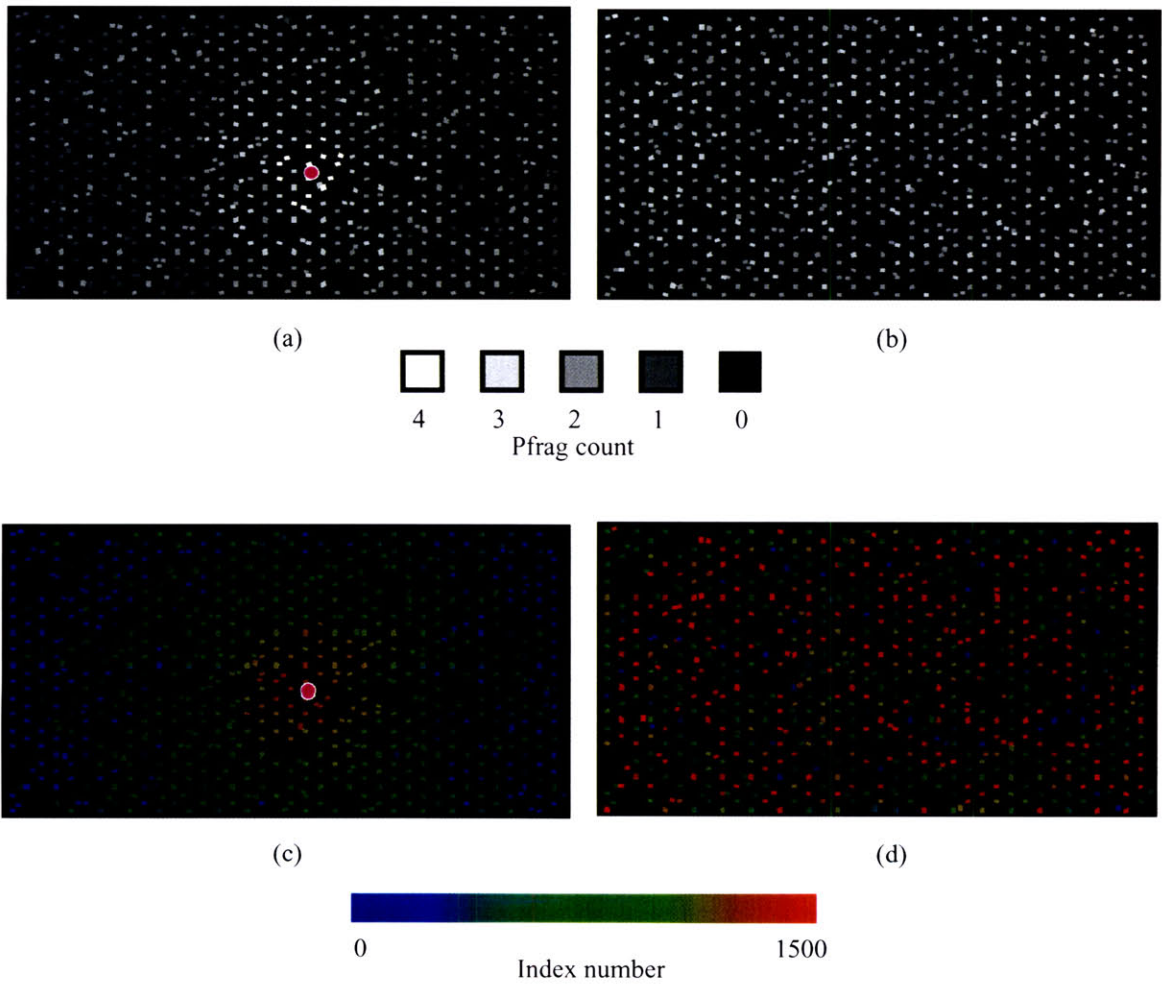
FIGURE 4-9. Diffusion: Transient and Steady State Behavior

Transient behavior: As the group of Diffusions stream in through I/O portal, the pfrag density falls off with distance from the portal (a). The Index number of the pfrag also varies smoothly with distance from the portal (c).

Steady State behavior: with streaming complete, the density homogenizes (b) and the position becomes random (c).

Channel Operator

The Channel operator uses three pfrags to construct a dedicated bidirectional communication channel between two I/O portals. Messages are transported as payload in carrier pfrags whose migration strategy uses the channel posts as guides. Channels position themselves dynamically. Multiple channels use short range gradients to inhibit (or at least discourage) overlap. Removal of either I/O portal triggers the a mass extinction of the pfrags.

The Channel operator employs three process fragments: Gradient, Tracer, and Halo. A channel is defined as an indexed sequence of Tracer posts spaced at pseudo regular intervals among the particles that separate the portals. Tracers enter the particle ensemble at the source portal, orient themselves to the Gradient from the destination portal and propagate to traverse the path between the portals. Halos likewise enter through a portal and propagate to build a sheath-like shell about the trail of Tracers (fig 4-10).

On entry into a particle, a Tracer writes to the HomePage a post consisting of 6 words (table 4-6). The ID's for the source and destination portals are constant throughout the channel. The hop count is determined dynamically. The Tracer fragment that originally enters the particle ensemble has the hop count set to zero. Subsequent copies of the Tracer have their hop count incremented by one, resulting in an incremental progression of hop counts along the trail of Tracers.

On calls to its *Update* handle, Tracer searches the local neighborhood for posts from other Tracers. In the equilibrium case, a given Tracer will find two posts with hop counts that differ from its own by +1 (the successor Tracer) and -1 (the predecessor Tracer). If no successor is found, the Tracer assumes that the path is still being grown. It then produces a replica of itself with a hop count equal to its own plus 1, and directs the replica to transfer to the neighboring particle that is the shortest distance from the destination. If no predecessor is found, the Tracer interprets this as a termination condition and calls its own *DeInstall* handle.

TABLE 4-6. Tracer post

Post Name						
Tracer	PfragID	Source ID #1	Source ID #2	Dest ID #1	Dest ID #2	HC

PfragID	the identifier common to all Tracers
Source ID #1	first identifier from source portal
Source ID #2	second identifier from source portal
Dest ID #1	first identifier from destination portal
Dest ID #2	second identifier from destination portal
HC	hop count = integer position of this pfrag in the sequence of pfrags which constitute the chain.

The Halo pfrag creates a near-field gradient radiating outward from the Tracers. Functionally, the Halo field is equivalent to the superposition of a collection of MultiGrad fields each radiating outward from a Tracer[22]. Table 4-7 shows the format of the Halo post. Once installed in an I/O portal, the Halo fragment propagates first to the particles occupied by Tracers and then disperses outward to the intervening particles. The range of the Halo is selectable. Unlike MultiGrad, Halo limits its propagation to those particle within the field range[23].

Fig. 4-10 shows a completed channel. A Gradient field from one portal is read by a second, which answers by sequencing the insertion of a Tracer and the Halo. The propagating Tracers walk the Gradient field back to its source. The Halo follows along, defining the channel's extent. Tracers from other channels treat this Halo as a repulsive force with a strength proportional to the proximity to the radiating Tracer.

---

22. The consolidation into a single pfrag saves space in both the HomePage and the RAM.

23. Recall that MultiGrad spreads to all the particles following the analogy of an omnipresent ether. Halo's limit their presence to those particles in range of the field. Like Gradient, Halo puts all its state data in the post. It would therefore be straightforward to create a "MultiHalo" vfrag as an additional efficiency.

TABLE 4-7. Halo post

Post Name							
Halo	PfragID	Source ID #1	Source ID #2	Dest ID #1	Dest ID #2	HC	Field Strength

PfragID	the identifier common to all Halos
Source ID #1	first identifier from source portal
Source ID #2	second identifier from source portal
Dest ID #1	first identifier from destination portal
Dest ID #2	second identifier from destination portal
HC	hop count = integer number of hops to the nearest Tracer.
Field Strength	real average of the proximal hop counts.
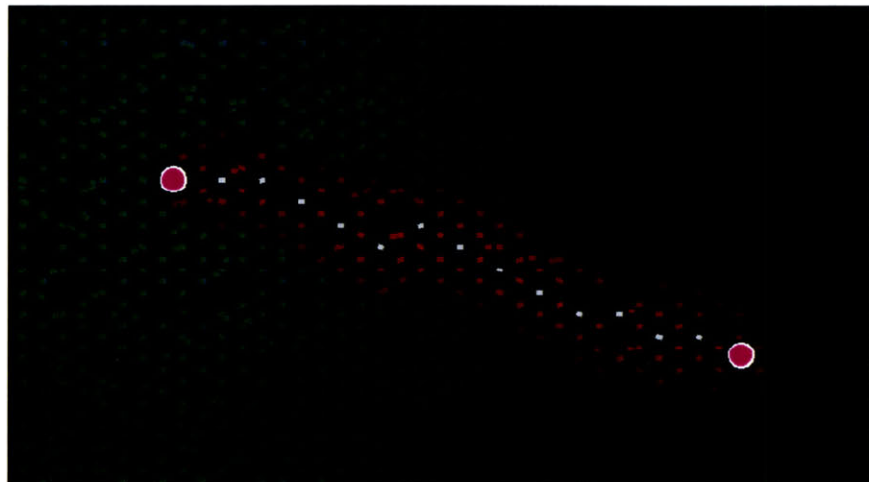
FIGURE 4-10. Channel connecting two portals

A Gradient propagates outward from a portal (green). A Tracer enters from the other portal and walks the Gradient field to its source (white). A Halo follows, its propagation guided by the trail of Tracers to form a sheath about the Channel

The routing of packet messages through dedicated paths is an elemental functionality for distributed systems. While practice will produce numerous variations on the basic theme, the simple Channel operator highlights characteristics that will be common to them all:

- **Bandwidth** The transmission efficiency of any channel is bounded by that point along the path where the communication between particles with Tracers is the slowest.

  The upper bound on signalling capacity is defined by the material characteristics of the medium in which the particles are suspended. Assuming that communicating particles can allocate that capacity as needed, the communication bandwidth between any pair of neighboring particles will naturally suffer when multiple pairs load the same patch of medium. The degradation will depend on how many pairs are competing for the local slice of bandwidth[24].

  This highlights the sensitivity to crisscrosses between the channels as well as a native problem with duplex service over a single Channel. In cases where bandwidth matters, 2-way means 2-Channel.

- **Latency** The most obvious scaling dependency is message latency. Increased channel length means more hops means greater latency. While the throughput remains constant, longer channels lengths, make the Channel operator less attractive for synchronous communication requiring intermittent acknowledgments.

- **Broadcast Excess** A secondary scaling effect derives from the use of a broadcast Gradient to orient the propagating Tracers. Gradients consume resources on every particle in the ensemble. And as the size of the ensemble grows, the cost of operating a Channel becomes disproportionately high. However, this is less a fundamental hurdle than it is an implementation bug which can be compensated for[25].

Channel is important as both a template and a proof of principle. While it is too rudimentary to be applied "as is", it is a useful template upon which additional features can be layered.

---

24. This situation is comparable to the case of multiple pairs of communication units competing for bandwidth on a contention based packet network such as the Ethernet.

25. Consider a variant of the Gradient with a 2 stage termination condition. One termination condition which expunged the pfrag everywhere except particles occupied by a Halo. The second condition being a general termination.

Coordinate Operator

The Coordinate operator uses two pfrags and a Channel operator to constructs a Cartesian coordinate system on a planar particle ensemble. Position is estimated relative to two I/O portals with known absolute position. The spatial extent is selectable, and dependencies are incorporated to void the operator's pfrags from the ensemble on command.

Operationally, copies of a Coord pfrag are deposited into all the particles in a selected region. These pfrags estimate the host particle's position using a combination of information that they carry and information that is read locally. The algorithm is a two stage estimation procedure (fig). Distances from the two I/O portals (d_1, d_2) are read from the local posts and combined with the distance separating the I/O portals[26] (d_p) to produce the partial estimate

$$[|x|, y]$$

The origin is portal #2 and the y axis lies along the line connecting the two portals. The distance from a third reference point is used to break the symmetry along the x axis and generate the final estimate.

$$y = (d_2{}^2 - d_1{}^2 + d_p{}^2)/2d_p$$

$$|x| = \sqrt{d_2{}^2 - y^2}$$

The third reference point is a Buoy pfrag that enters the particle ensemble through a portal and positions itself relative to the fields from the I/O portals. It computes its position[27] and radiates a field with the location and ID data embedded. Note that distance estimates relative to the Buoy are doubly influenced by the bounded accu-

---
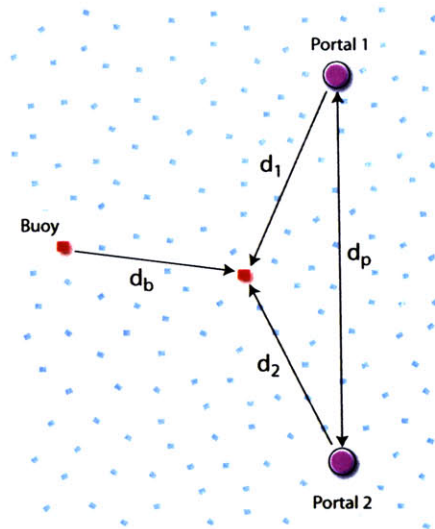
26. d_p is embedded in the Coord pfrag at the originating I/O portal.

27. Uses the positioning algorithm and assuming the positive value for x

racy of the gradient fields. Use of the Buoy field is therefore restricted to symmetry breaking.

The Coordinate operator combines the Channel operator together with two new pfrags; a Buoy and the Coord. The coordinate system is constructed by sequencing the insertion of the pfrags through the two I/O portals[28]. On contact with the ensemble, each portal radiates a Gradient. portal #1 reads the estimate of distance from portal #2 and stores this value for insertion into outgoing pfrags. Portal #1 issues a Buoy that migrates toward a preferred position, radiating a MultiGrad field on arrival. Portal #2 then issues a Channel operator which walks the Gradient field back to portal #1. The Halos from this Channel serve as an upper bound on the extent of the coordinate system[29]. On arrival of the Channel, portal #1 then issues a Coord pfrag which spreads using the Halo as a propagation guide. Once installed, each Coord estimates the position of its host particle and posts this estimate to the local HomePage (table.4-8).

TABLE 4-8. Coord post

Post Name								
Coord	PfragID	Enable Flag	X	Y	Source ID #1	Source ID #2	Dest ID #1	Dest ID #2

PfragID	the identifier common to all Coords
Enable Flag	boolean to indicate that the computed position of the pfrag is within a prespecified range..
X, Y	Position estimate (relative to portals).
Source ID #1	first identifier from source portal
Source ID #2	second identifier from source portal
Dest ID #1	first identifier from destination portal
Dest ID #2	second identifier from destination portal

---

28. The requisite coordination between the portals can be maintained either through a external link or by using the Gradient ID information to assume roles and sequencing their activity on the arrival of pfrags propagating outward from the other portal.

29. Analogous to the use of a "primer coat" of paint to define a region and prepare it for the ensuing layers.

Two of the stages of this process are shown in fig. 4-11. In fig. 4-11a, the Buoy is in place and gradient fields are radiating from both the I/O portals and the Buoy. In fig. 4-11b, the Coords have been dispersed over the selected region and have posted their coordinates.

The limitations of the Coordinate operator derive from its sensitivity to noise in the gradient-based distance estimates, and by extension, to the density of the particles. The symmetry breaking scheme involving the Buoy is similarly error prone, with the result that the sign bit for x in the range $-0.5 < x < 0.5$ is unusably noisy for thresholding on $x = 0$.

The import of the Coordinate operator extends to several domains. In general, planar surfaces with a highly resolved coordinate reference would have wide application outside the domain of self-assembly. For example, small devices can be placed on a table and use their table-relative coordinates to define their interaction in position-dependent ways.

For self-assembling systems, a Coordinate operator affords an ideal scaffold for local orientation of migrating pfrags, which themselves can constitute the building blocks of more complex structures. Support for both limited-extent and overlapping coordinate systems broadens the application domain even further.
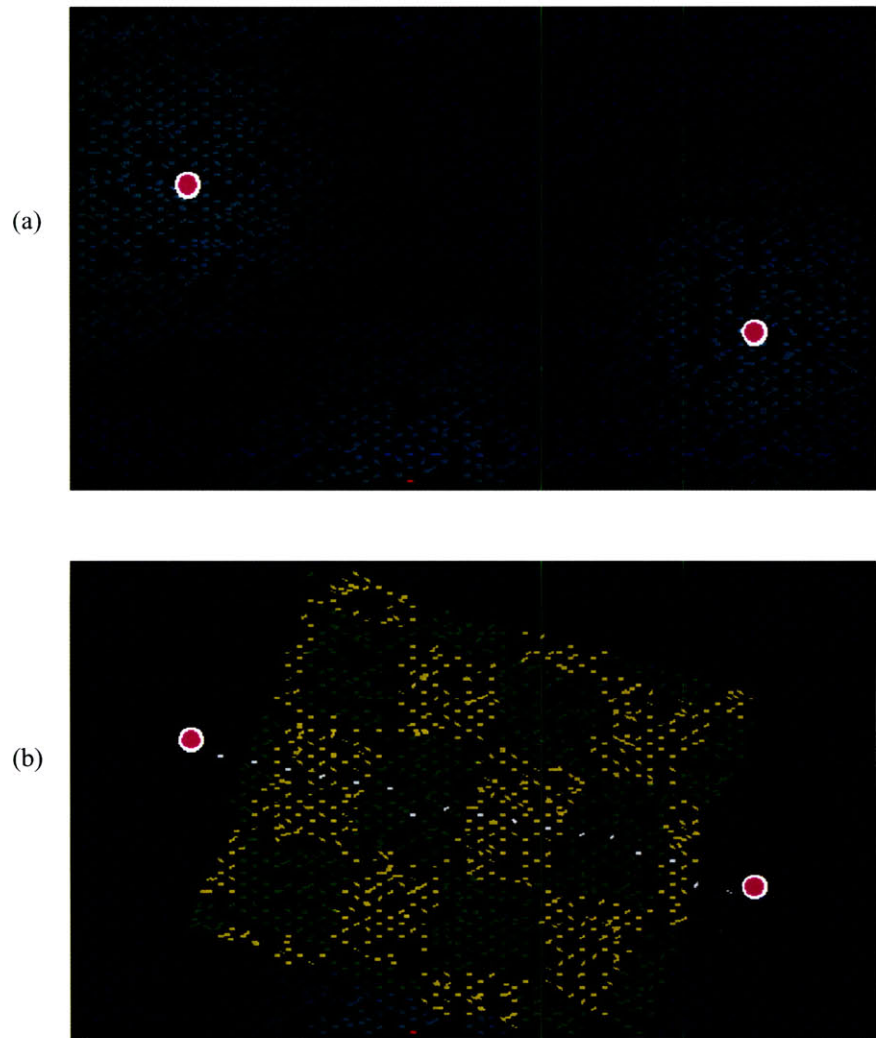
(a)

(b)

FIGURE 4-11. Construction of Coordinate System

Two external portals radiate Gradient fields. The fields guide the positioning of a Buoy (red particle bottom center) which itself radiates a third field (a). A Channel operator defines a region between the two portals. Coord pfrags blanket this region and interact to estimate their 2D coordinates relative to the portals.

Summary

Process fragments (pfrags) are the reactants that fuel self assembly on a *paintable*. The definitions of chapter 3 admit pfrags with a wide variety of size, complexity, and behaviors. This chapter presented a detailed description of six representative pfrags. These six examples both illustrate the principles of pfrag design, and form a foundataion pfrag library for the applications of the next chapter. Table 4-9 summarizes their functionality and lists references to related work.

Two forms of self-assembly were demonstrated: scaffolded and thermodynamic. Both produced structures that were stable, adaptive and in bounded compliance with a set of prescribed global criteria.

Self-assembled program structures are inherently adaptable. Pfrags insure this by perpetualizing the generative processes that gave rise to the structures in the first place. For example, in thermodynamic assembly, the component parts of a structure are positioned in accordance with a minimization function applied to some environmentally dependent variable(s). In effect, the structure is an equilibrium state of an energy functional. In a system that is constantly checking these input variables and recomputing the minimization, adaptation is a natural byproduct. As long as the environment remains stable, the structure should too. Changes in the environment are reflected in the automatic progress toward a new minimum[30]. The degree of change can range from small perturbations to localized extinction followed by regeneration from the survivors[31].

Abstraction and modularity are cardinal elements of program design. This chapter introduced the constructs of a vfrag and an Operator as prototypical examples of abstraction and modularity in process self-assembly. Any pfrag that includes all of its state in its HomePage posts is a candidate for emulation by a vfrag. This makes vfrags an attractive programming artifice in applications where the particle RAM space is a limiting resource. Operators are collections of pfrags organized after the metaphor of biological cells, each specializing and clustering to form "organs".

---

30. In a centralized architecture, this excess would be difficult to justify. It would amount to binding the value of variable **Z** to the sum of **X** and **Y**, and then continually re-adding **X** and **Y** just to insure that **Z**'s value is current.

31. consider an example from the Channel operator. If a particle containing a Tracer pfrag dies, the downstream Tracers have their dependencies broken and *DeInstall*. The Tracer just upstream of the fault assumes that it is still in growth mode and propagates a new downstream Tracer in an identical manner to when the Channel was initially grown.

TABLE 4-9. Summary: six examples of pfrag self assembly

Component Name	No. of pfrag types	Description	Related Work
simple pfrags			
Gradient	1	For each particle, Gradient estimates the shortest distance back to a fixed anchor point — usually an I/O port. Gradients spread virally and interact locally to build fractional distance estimates. Resolution limited by particle density. Perhaps the simplest of all pfrags. Smallest too (appr. 500 bytes).	[28] [37] [40] [43]
Diffusion	1	Basic "carrier" pfrag. Each Diffusion accepts a data packet as a payload. Once in the particle ensemble, interacting Diffusions seek to position themselves to minimize disparity in local pfrag density. This drives a continuous shuffle that decorrelates the pfrag's position in both space and time.	course texts[a]
vfrags			
MultiGrad	1	Archetype "virtual pfrag". One Multigrad can emulate the behavior of multiple Gradients. Arbitrary pfrags can radiate fields over a selectable radius by simply posting a "center point" to the local HomePage. This center point is read by MultiGrad which then channels the field.	specific to this programming model
Operators			
Tessellation	2	Groups the particles into Voronoi regions about a set of uniformly distributed anchor points. Uses two pfrags. MultiGrads channel fields radiating from mobile Centroids. Centroids interpret neighboring fields as repulsive spring force and adjust their spatial position to minimize free energy.	course text[b]
Channel	3	Defines a bidirectional communication channel between two anchor points. Uses three pfrags. A stream of Tracer pfrags propagate a trail from one anchor following the Gradient field radiated by the other anchor. Halo pfrags build a mini-gradient field centered about each of the Tracers, effectively forming a repulsive sheath around the channel	[14] [37] [43] [44]
Coordinate	5	Constructs 2D coordinate system from two anchor points. Uses Channel operator plus two additional pfrags. Buoy breaks axial symmetry of anchors. Coordinate pfrag spreads over region defined by Channel and triangulates off of Gradients from two anchors and Buoy.	[37] [41]

a. Diffusion is seldom applied directly in distributed computing systems. For analogous physical behavior, see work on diffusion limited aggregation and techniques for mathematical modeling [20].

b. Also seldom applied directly in distributed computing, See the treatment of *Thermodynamics* from intro physics texts

Operators are a weaker form of program abstraction in that their inner workings are partially visible[32]. Yet, an Operator can present a well-defined interface to the external pfrag environment, independent of the Operator's inner workings. A case in point is the Coordinate operator, where the external interface is the Coord posts.

Finally, this chapter explicitly extended the notion of inter-pfrag signalling. In chapter 3, signalling between pfrags was limited to posts placed on mutually viewable HomePages. This naturally limited the range over which pfrags could communicate to a single communication radius. This chapter described the device of a carrier pfrag which accepts a message as payload and implements a migration strategy which (ultimately) transports the messages to its destination. As a solution to messaging congestion, the Channel operator was described as a means for dynamically creating a structured path for the carriers.

---

32. Signalling between the Operator's pfrags is via HomePage posts which are externally visible.

Applications

The system is optimal, we just don't know what for.

- Gerald Jay Sussman   (remark at a group meeting)

This chapter combines the foundation pfrags of chapter 4 to simulate four representative applications: Audio Streaming, Holistic Image Storage, Surface Bus and Image Segmentation.

In the first two applications, the particle ensemble functions primarily as a single unit of continuous memory. Use of the ensemble's native compute ability is limited to dynamic positioning of the data, and — in the case of Holistic Data Storage — signal processing for frequency domain transformation. The Surface Bus application explores peer-to-peer communication between devices that are arbitrarily positioned on the periphery of a table. Peers are defined as any computational artifact that can exchange data; laptops, PDA's, keyboards, displays, etc. The particle ensemble — embedded in the surface of the table — acts as both a communication medium and as a computational resource for these devices. In the Image Segmentation application, I/O to the particle ensemble is extended to include parallel I/O from a dense array of photo sensors embedded among the particles. Sensors sample the image on an irregular raster and then propagate the pixel data to the neighboring particles via near-field MultiGrad gradients. A popular algorithm for supervised classification is implemented as a competition among "expert" pfrags, each selective for a particular image attribute.

Collectively, the applications demonstrate support for storage, communication, and signal processing. More significantly, they all point to practical instances where the paintable architecture and process self-assembly scale beyond the operational complexity limits of contemporary computing systems[1].

Streaming Audio

This section considers the general problem of streaming media data on a *paintable*, with a focus on audio as illustrative example. The important subproblems are the transmission of the data to and from the particle ensemble, the storage characteristics of the ensemble, and the performance of the retrieval. Mass storage on a *paintable* is complicated by the fact that there is no functional equivalent to a global address generator. It is therefore incumbent on the data packets themselves to "know where to go" — and to make this decision dynamically in response to the instantaneous topology of the particle ensemble.

The goal of this application is to store 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

Representation. The solution advanced here is to encapsulate each audio packet in a separate Carrier process fragment. During pre-processing, analog audio is sampled and diced into packets (fig. 5-1). Each packet is embedded into a Carrier as payload. 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 *Update* cycle, each Carrier selects one of two modes; the diffusion mode for storage or the call-back mode for retrieval[2].

---

1. As distinct from the formal measures of algorithmic complexity, the phrase "operational complexity" is used as an informal reference to the cost of constructing and maintaining an engineered system — component purchase, design, manufacture, test, support, service, and liability insurance.

FIGURE 5-1. Representation of audio for use in a *paintable*.

The analog waveform (*a*) is sampled and organized into packets (*b*). Each packet is enveloped by a pfrag (*c*) along with descriptive data for the packet. Embedded in the pfrag is a migration strategy that directs the pfrag's motion within the particle ensemble.
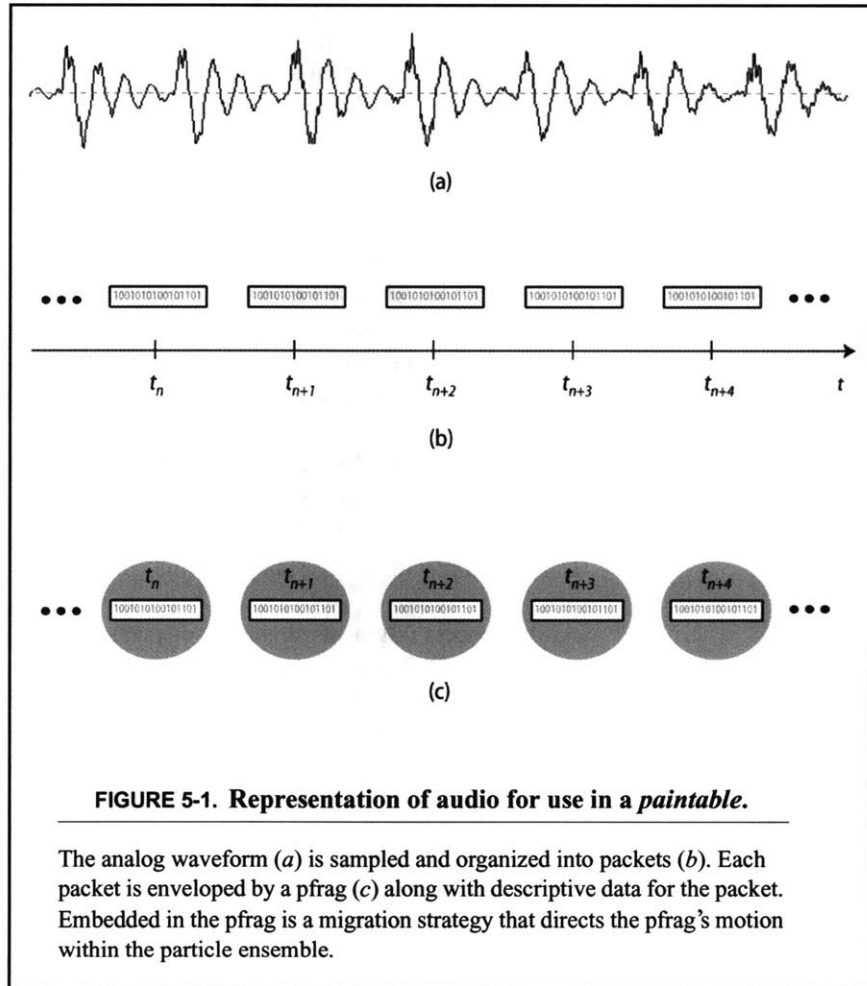
Storage. In the diffusion mode, the emergent behavior is the quasi-uniform distribution of the Carriers over the entire particle ensemble, and a spatial shuffling of the Carriers. The rule set for this mode is identical to that described in chapter 4 for the Diffusion pfrag. This is the default mode and is active during storage.

---

2. Each of these modes is defined by a simple rule set encoded in the Carrier's *Update* handle

Retrieval. Carriers enter their call-back mode when they see posts from a Call-BackGradient pfrag. On contact with the particle ensemble, an output portal[3] inserts a CallBackGradient pfrag which radiates a gradient field. The posts from this gradient field contain estimates for the distance back to the output portal, an ID for the requested audio stream, and a range of time codes which are 'active'; i.e. those that should soon be queued for playback. When a Carrier sees the post from the CallBackGradient, it does one of three things;

1. If its time code falls within the range of active time codes, it proceeds directly toward the output portal.

2. Else if it too close to the portal, it moves away from the portal, thus increasing the bandwidth efficiency in the vicinity of the portal.

3. Otherwise, it builds a local average of time codes, and adjusts its position relative to this average using the gradient field for orientation.

Experimental Results. The performance was examined on the simulator using a configuration of 660 particle, each communicating with 10 to 16 of its immediate neighbors. A segment of audio was divided into 1500 packets, each embedded in a Carrier along with a corresponding time code[4].

As the process fragments stream in, dispersive pressure tends to impose a spatial order (fig 5-2). A process fragment's distance from the input portal tends to be proportional to the order in which the process fragment entered the ensemble, with the earliest entries being furthest away.

---

3. The "Output portal" is part of an external device that is requesting audio data.

4. Apologies to the alert readers. Yes this is indeed the same simulator configuration used to demonstrate the basic operation of the Diffusion pfrag. Strictly as a convenience, the author used the figures from this streaming experiment in the write up for Diffusion.
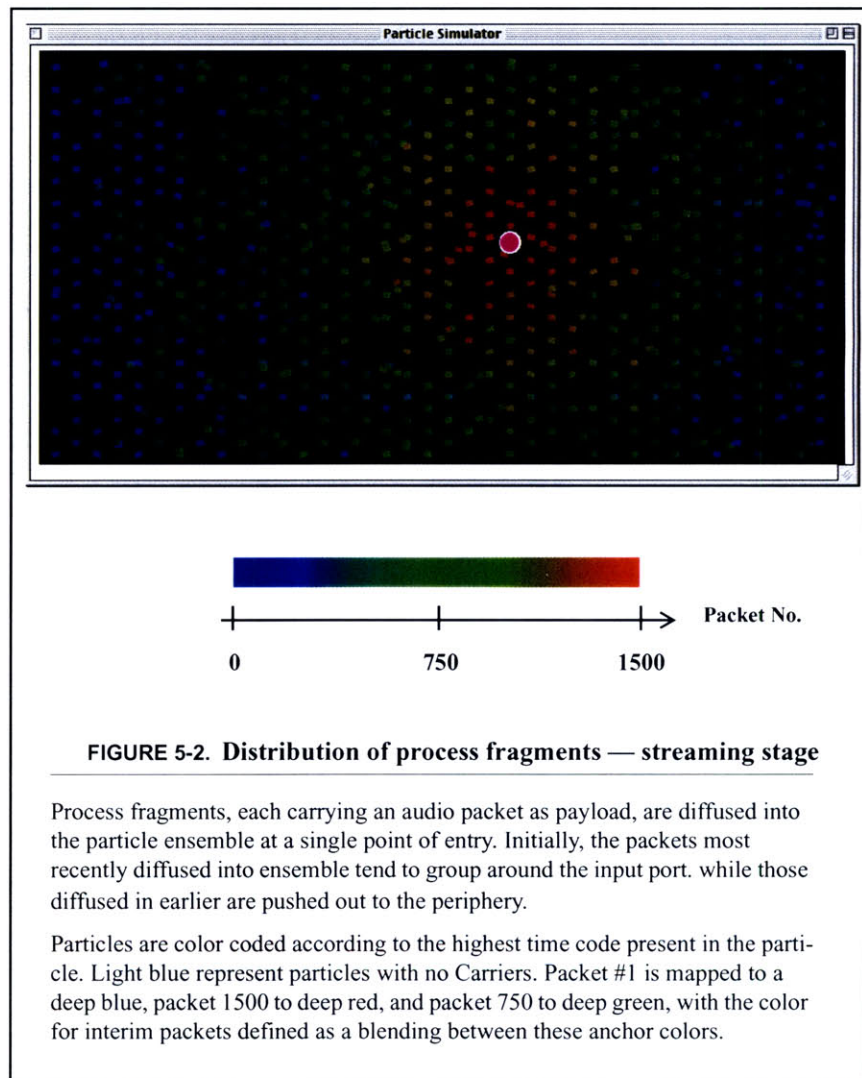
FIGURE 5-2. **Distribution of process fragments — streaming stage**

Process fragments, each carrying an audio packet as payload, are diffused into the particle ensemble at a single point of entry. Initially, the packets most recently diffused into ensemble tend to group around the input port. while those diffused in earlier are pushed out to the periphery.

Particles are color coded according to the highest time code present in the particle. Light blue represent particles with no Carriers. Packet #1 is mapped to a deep blue, packet 1500 to deep red, and packet 750 to deep green, with the color for interim packets defined as a blending between these anchor colors.

Once the last of the fragments is in, outward dispersive pressure gives way and the fragments seek to uniformly distribute themselves throughout the memory. Each Carrier selects its next transfer destination with an eye toward minimizing any disparity in the local density. The end effect is that global distribution of Carriers tends toward uniformity and the spatial ordering of the Carriers is randomized.

FIGURE 5-3. Distribution of process fragments — steady state

Ultimately, after all the code segments have been diffused into the particle ensemble, the diffusive mechanism dominates and the position of the pfrags becomes randomized.
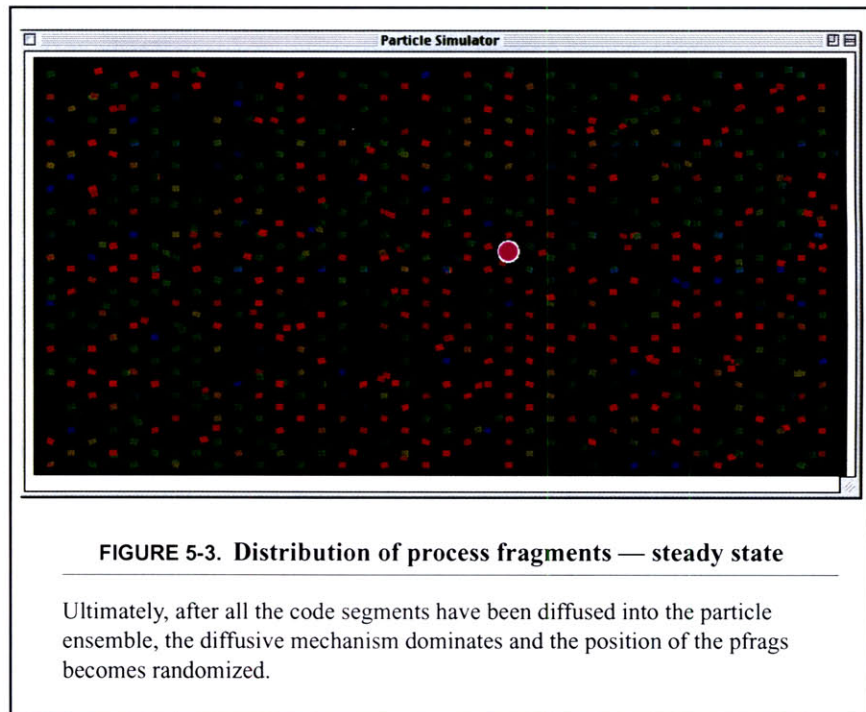
Retrieval is initiated when an output portal makes contact with the particle ensemble and radiates a field of CallBackGradients. The posts from the CallBackGradient specify a range of active time codes, which the I/O portal updates as the streaming progresses (fig. 5-4). If a Carrier finds that its time code falls within that range, it proceeds directly toward the output port. Otherwise, the Carrier repositions itself so as to restore a spatial ordering where the Carrier's distance from the portal is proportional to its time code.
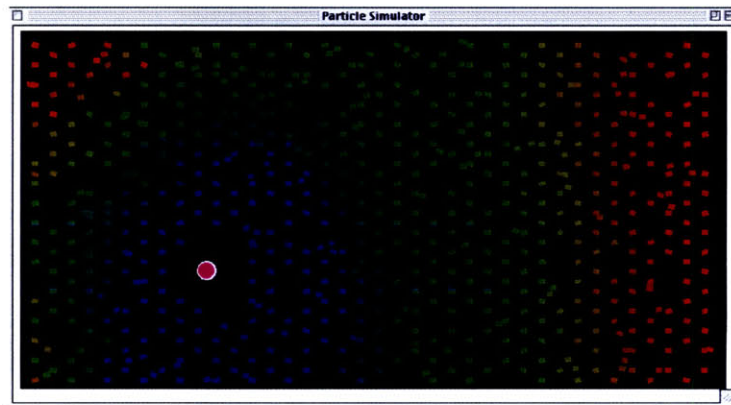
(a)
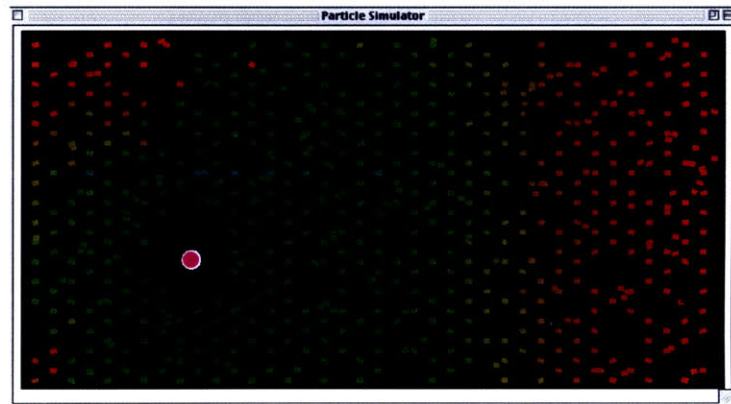
(b)

FIGURE 5-4. Retrieval

early stage (a) An newly positioned I/O port first radiates a "call back gradient" (not visible). Shortly afterward, the carriers begin to align themselves into concentric rings, with radial distance increasing with the time codes.

late stage (b) As the carriers arrive at the I/O port and deliver their packet payloads, the remaining carriers reposition themselves relative to the output port and the port issues updated range values for the active time codes.

Discussion. This application is a modest extension of the Diffusion pfrag. And yet it is noteworthy that an application can be built around even so simple a pfrag and still support useful functionality. In the case of streamed storage of audio, the attractive properties are:

- **shuttle mode playback** At any arbitrary position in the ensemble, a local region of suitable size will contain samples of the entire audio stream taken at pseudo random intervals.

- **ubiquitous table of contents** At any arbitrary position in the ensemble, the local region will contain pfrags from all the audio sequences currently stored in the particle ensemble.

- **fault tolerance** Failure of all the particles within an arbitrary closed region will result in sporadic dropouts during play back. Such drop out can often be partially masked or corrected[5].

- **no topology dependence** Pfrags will evenly distribute themselves over any collection of particles which can be described by a connected graph — regardless of the graph's topology.

On a more basic level, this application gives a first indication of the power of this architecture to challenge convention — in this case, the belief that networks need routers. Although much of the audio packet's behavior involves what we traditionally regard as networking, this system is absent of any dedicated routing device. Control over channeling and transport which typically has been vested in separate routers has been exclusively deeded to the behavior emerging from local interaction among packets in a strictly homogeneous computing environment.

---

5. This is similar to the popular data shuffling and strifing method of error protection employed on consumer audio CDs

Holistic Data Storage

In this application, we again treat the collective memory of the ensemble as a monolithic slab of memory, but this time with the unique characteristic that it stores images and sound holistically - arbitrary subsets of the memory yield reduced images of reduced resolution, in a manner reminiscent of a hologram.

Essential Holistic Storage. The concept is comparatively new to the digital storage community. Fig. 5-5 summarizes the essentials. Given a digitized image and a memory element embodied as a 2D planar surface, the goal is to select a representation for that image such that it can be stored into the memory plane. If the contents of the memory are read out and decoded, the reconstructed image should be identical to the original ... yawn. However if the memory surface is arbitrarily fragmented in such a way that the majority of the stored image data is lost, leaving only a fraction of the data available, the decoded images should be low resolution versions of the originals. In other words, loss of data should manifest itself as loss of sharpness.

Several techniques for holistic image representation are available [8]. This example employs a cascade of hierarchical wavelet transformation, duplication of the lowest frequency transform coefficients, and a pseudo-random 'diffusive' scattering of the transform samples throughout the memory. The net effect is that small sub-areas of the memory are more likely to contain a nearly complete complement of the lowest frequency subbands, which ensures at least a blurred image on reconstruction.

However, as the size of the memory patches decreases, there is an increased possibility that the recovered data will be missing some samples of the low frequency data, resulting in holes in the reconstructed image (fig. 5-5e). Small concentrations of these holes can be compensated for by interpolation. But as the memory patch size continues to shrink, the image just breaks apart (fig. 5-6). Given an undersampled image, data from additional fragments of the memory add constructively to yield a progressive refinement.

Implementation. On a *paintable*, holistic image storage is implemented as the interaction between two pfrag types: Carriers and Transforms. Prior to entry into the particle ensemble, an image is divided up into blocks. Each block is embedded in a separate Carrier along with descriptive parameters such as the image-relative coordinates. The Carriers are streamed into the particle ensemble where they spread via diffusion.
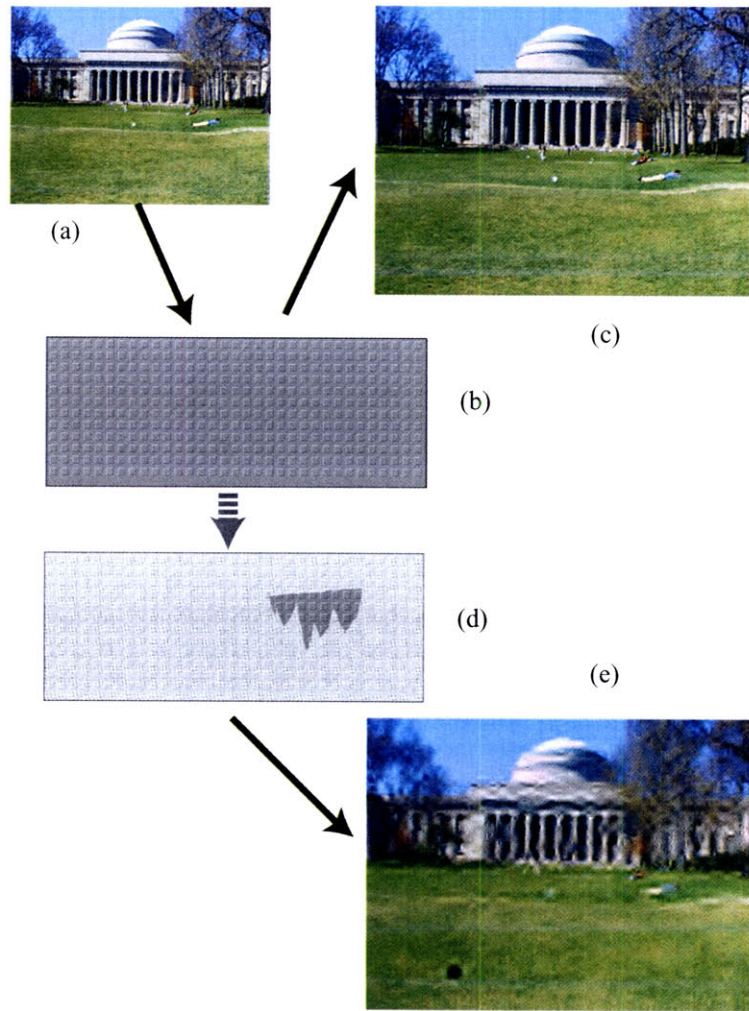
FIGURE 5-5. Essential holistic image storage

An original image (a) is sampled, encoded and stored in memory (shown in (b) as 2D planar surface). The original image can be reconstructed from the complete contents of the memory (c). However, when the memory element is damaged and only a small portion of the originally stored data is available (darkened portion of (d)), the reconstructed image is a low resolution copy of the original.

FIGURE 5-6. Essential holistic image storage: progressive refinement

An example of progressive refinement using images taken from preliminary simulations of holistic image storage. "Holistically" encoded image data is mapped to a rectangular array of memory. Lossless reconstruction of the image requires recovery of the data from the entire memory array.

Data extracted from a subsets of the memory array (darkened areas of (a)) decode to an image (b) that is a random sampling of the space and spectrum of the original. Additional fragments of the memory ((c) and (e)) combine constructively to yield a progressive refinement of the reconstructed image ((d) and (f))

A single Transform process fragment is streamed into the ensemble and propagates a 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 (fig 5-7). The output of this transformation is 10 subbands, each of which isolates energy at a selected orientation and frequency [6]. 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. The 8x replication of the lowest frequency subband corresponds to a 12.5% signal redundancy [7] for storage.

Experimental Results. Experiments were conducted with the simulator configured for 1200 particles. Two color images with dimensions 320 x 240 are broken up into blocks with dimensions 32 x 24 pixels, inserted into Carrier pfrags and diffused into the particle ensemble where they interacted with the Transform pfrags and spawned the mini-Carriers (fig. 5-8). The positioning and the timing for the insertion of these two images were arbitrarily displaced. With the input streaming operation complete and the density approaching the steady state, a subregion was

---

6. This application admits a variety of frequency transformation techniques. The two-tap Haar kernel [+1 -1] is the simplest example of a transformation that is both reversible and complete. Transforms are reversible if the original image can be losslessly re-synthesized from the complete set of transform samples. Transforms are complete if the total number of transform samples is identically equal to the total number of pixels in the image. For an excellent general reference, see [46].

The 1D kernel is separably applied in 2D to produce four subbands — subsampled, band-passed versions of the original image data. Each of the four subbands has dimensions equal to one half of the dimensions of the original image block. One subband is a low frequency subband, corresponding to a 2x decimated version of the image. The remaining three high frequency subbands contain edge detail oriented in the horizontal, vertical and diagonal (respectively).

At the first two levels of the three level pyramid, the entire transformation is recursively applied to the low frequency subband from the previous level. The original image block is therefore losslessly represented by 10 subbands, three high frequency subbands from each of three levels, plus the low frequency subband from the last level.

7. The low frequency subband contains $1/64^{th}$ the size of the original image.
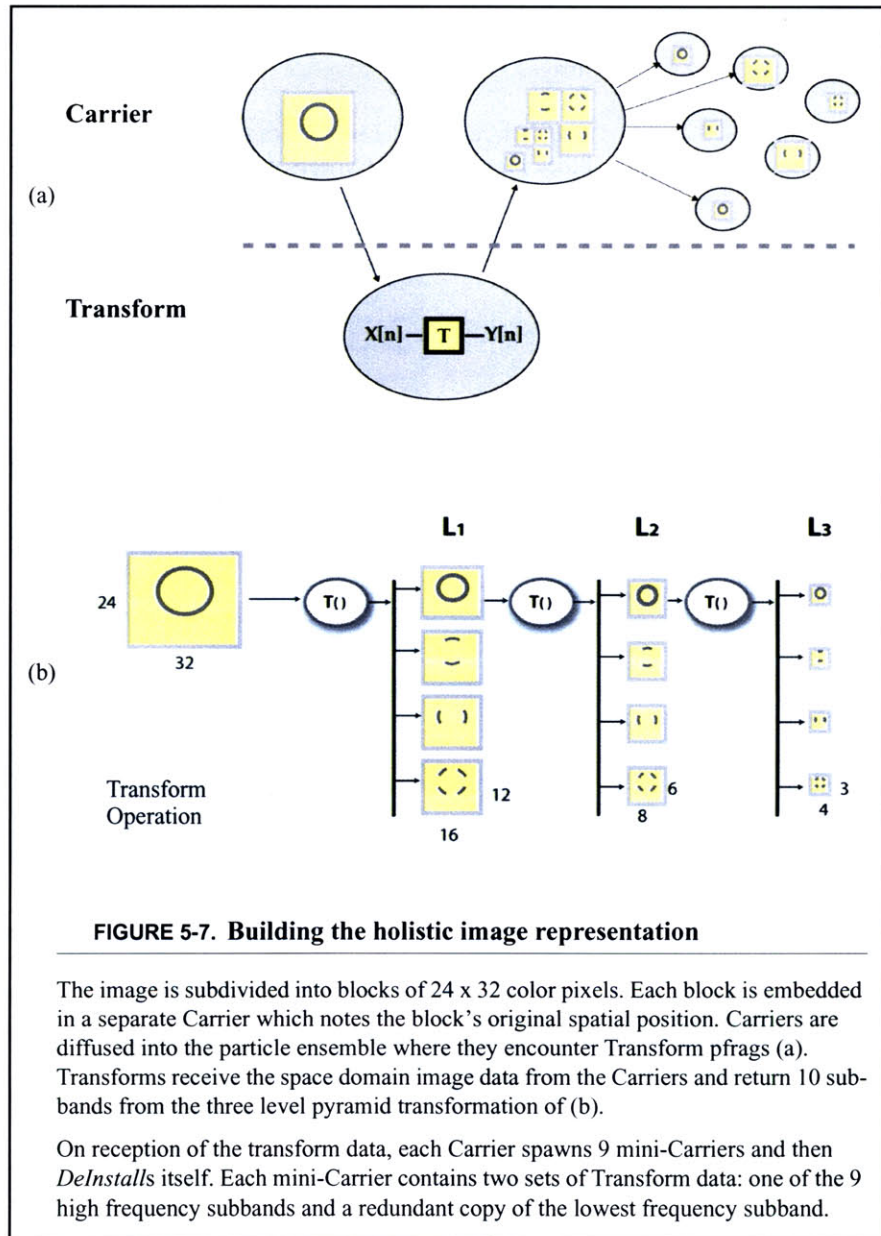
FIGURE 5-7. Building the holistic image representation

The image is subdivided into blocks of 24 x 32 color pixels. Each block is embedded in a separate Carrier which notes the block's original spatial position. Carriers are diffused into the particle ensemble where they encounter Transform pfrags (a). Transforms receive the space domain image data from the Carriers and return 10 subbands from the three level pyramid transformation of (b).

On reception of the transform data, each Carrier spawns 9 mini-Carriers and then *DeInstalls* itself. Each mini-Carrier contains two sets of Transform data: one of the 9 high frequency subbands and a redundant copy of the lowest frequency subband.

selected and the Carriers occupying that subregion were diffused out through a single output portal (fig. 5-9). The payload data from the Carriers was extracted, synthesized back to the space domain pixel blocks, and reassembled back into the images.

The dark rectangular regions of the reconstructed images correspond to image blocks for which no Carriers were present in the selected subset of particles.

FIGURE 5-8. Holistic Storage: Streaming input

Snap shot of two images diffusing into the particle ensemble from separate potals.

Images (b) and (c) with dimensions 320 x 240 are divided into blocks, inserted into Carriers pfrags, and streamed into the particle ensemble (a). Particles are color coded to reflect the number of Carriers they contain, with increasing gray level corresponding to more Carriers (blue particles contain no Carriers).

The images are inserted into the particle ensemble at different times. Streaming of image (c) through the portal on the lower right is almost complete and the Carriers have largely dispersed. Streaming of image (b) through the portal on the upper left has just begun and the Carrier concentration near the portal is still high.

FIGURE 5-9. Holistic Storage: sample output

An arbitrary subset of the particle ensemble was selected and the Carriers were diffused out through an output portal. The transform data was extracted from the Carriers and decoded to recover the image blocks.

The particle ensemble (a) is shown during the read out. Blue are particles already void of Carriers,. The distribution of the remaining Carriers is shown by the gray-levels.

The two images (b) and (c) are the cumulative results of reconstructing all the Carriers after read out is complete. Black areas of images correspond to blocks for which no low frequency data was available.

These simulations confirmed several essential characteristics:

- The position of the image packets can be successfully decorrelated from the spatial organization of the particles. In other words, nothing in the shape of the selected subset of particles suggests the positioning of the dropouts in the reconstructed images.

- Recovery of additional packets results in a progressive refinement of the reconstructed images.

- The technique scales upward to include multiple sources from multiple I/O portals.

Discussion. A few additional points are worth noting:

- Passing image data on the HomePage was a disaster. It is slow and unnecessarily loads the networking subsystem with useless updates. This application makes clear that the programming model will have to be amended to support a more direct mode of data transfer between pfrags which are co-located in a particle. The expectation is that the setup and handshaking will be comparatively low bandwidth and can proceed through HomePage posts. Support for a direct memory-to-memory transfer between pfrags will likely be a natural extension to the OS supported Pfrag Toolkit.

- Additional intelligence can be incorporated into the Carriers to enhance adaptability. For example, while the memory capacity of any particle ensemble will be limited, this inherently hard limit can be made to appear soft. In saturation, the Carriers can cooperate to eliminate those carrying the high frequency subbands as a means for making room for additional incoming images. This creates the illusion of a memory with a tiered set of limits.

- The range of data types suitable for holistic representations can include some non-obvious examples. Consider bank account data. One might initially consider a financial portfolio with data such as the number of accounts, specific balances, and service authorizations to be ill suited for a hierarchical representation. But this data can in fact be categorized at multiple levels. For example, at the coarsest level, an individual might be thought of as *rich, poor,* or *middle class.* Subsequent levels of refinement would create an increasingly detailed picture of the individual's net worth and financial activity. While typical transactions require the most detailed information available, some in fact do not (an ATM can safely allow the purchase of $30.^{00} worth of gas by some one who was 'rich' last night — with a blackout zone centered at Las Vegas).

Surface Bus

Imagine an instance of a tabletop populated with a collection of randomly positioned objects and where the table itself provided the network connectivity. Whenever a new object is placed onto the table, the table automatically adapts the network configuration to incorporate the new device. Yet in contrast to similar scenarios involving near-field RF, transmission paths through the table also supply computation capacity as a byproduct of the message passing. Devices positioned on the table pass both messages and coded procedures to apply to those messages. This section describes "Surface Bus" — a technique for employing a paintable computer to jointly supply capacity for both transmission and computation, and to afford the external devices some ability to adaptively reconfigure these resources as needed.

Problem Domain. The target platform is a 2D particle ensemble embedded in a planar surface such as a conference room table. A typical example would be a sheet of plywood with the ensemble of computing elements laminated into one of the layers. Over the course of a meeting, multiple compute-enabled devices (laptops, PDA's, cameras, etc.) will be randomly placed on the periphery of the table. These device will often wish to communicate and share data. In contemporary scenarios involving near-field RF, the devices are integrated into pico-nets built on short range wireless links. Membership in that network requires a single transceiver device drawing enough power to speak to the most distant point in the local network (nominally 2 or more meters). In the alternative case of a *paintable*, external devices would be fitted with (perhaps multiple) transceivers similar to those on the *paintable* particles. These transceivers, positioned on the bottom of the device, would make contact with those particles in their immediate vicinity, essentially assuming the role of portals between the particle ensemble and the external device.

In this scenario, the two natural questions are:

1. Can we employ the ad hoc communication between the particles to support communication between the external devices?
2. Can we take advantage of the compute capacity inherent in the particles which pass messages to actually perform directed computation on the data as well.

Approach. The approach taken in this section is to partition the particle ensemble into two non-overlapping regions. The particles located near the periphery of the table form a communication-only region. The remaining particles in the table s interior constitute the second region that jointly supports communication and ancillary processing. Devices placed on the tabletop assume the role of peers in a multi-hop communication network. These peers use Channel Operators to build links to

selected neighboring peers. The links, in turn, aggregate into a ring network positioned near the periphery of the table. The particles belonging to this ring network delineate the communication-only region. The particles enclosed by the ring form the second region, where particles perform directed processing in addition to communication. Peers treat the particles in this second region as a shared, reconfigurable processing resource. Once access to this resource is negotiated, any two peers can initiate processing using a suitably augmented Channel Operator.

The presentation in this section is organized as a series of building blocks. Peers are defined as a collection of portals arranged in a fixed geometry. The Buoy pfrag is introduced and its role in channeling is illustrated. Peers, augmented with Buoys, use Channel Operators to grow peer-to-peer links. The assembly of links is sequenced to construct an adaptive ring bus. Finally, a method for constructing secondary links capable of concomitant processing is described.
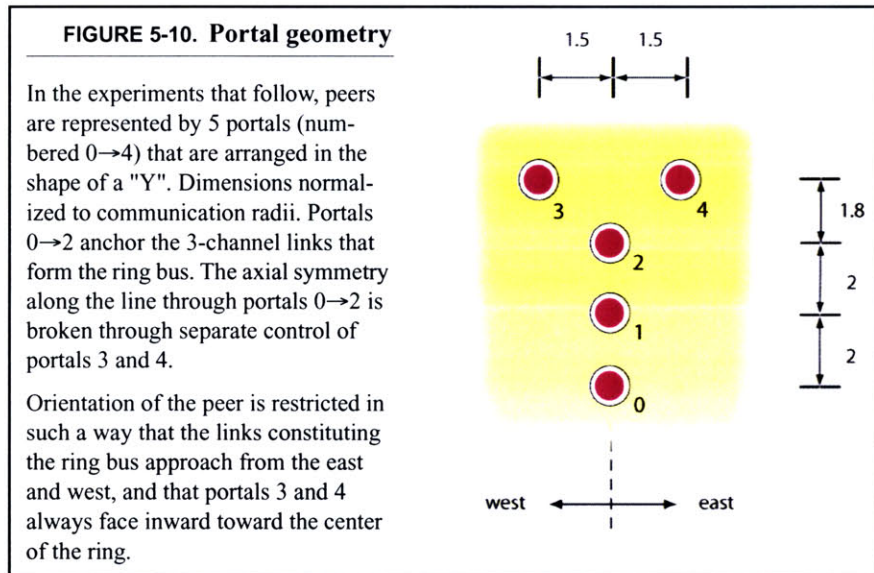
Single Peer. The elemental building block of a Surface Bus network is the peer. Randomly positioned along the periphery of the table, each peer establishes direct, multi-channel links with one or two of its spatial neighbors, thus forming a multi-hop network organized in a ring topology[8]. Peer devices are addressable via unique, device-specific IDs that are either pre-stored at time of manufacture or dynamically assigned. On contact with the table, a newly introduced peer issues a signature Gradient[9], senses the relative position of the existing peers, and signals to its immediate neighbors in order to negotiate its role in construction of a multi-channel link.

Surface Bus peers are defined as a group of portals arranged in a suitable geometry. Materially, the portals will typically be low power transceivers embedded in the bottom of the peer device's outer casing. Four criteria define a geometry as "suitable":

1. A footprint that is small enough to accommodate portable, pocket-sized devices.
2. Any peer can estimate the relative position of all the other peers based on signature Gradients that the other peers emit.
3. Links with up to two neighboring peers can be constructed from Channel Operators that do not overlap.
4. There is likewise an unobstructed path between any two peers (not just those that are spatially proximal) that can be used to open a temporary connection.

---

8. The ring can be open or closed
9. A "signature Gradient" is one whose post contain information suitable for identifying or characterizing the source.

FIGURE 5-10. Portal geometry

In the experiments that follow, peers are represented by 5 portals (numbered 0→4) that are arranged in the shape of a "Y". Dimensions normalized to communication radii. Portals 0→2 anchor the 3-channel links that form the ring bus. The axial symmetry along the line through portals 0→2 is broken through separate control of portals 3 and 4.

Orientation of the peer is restricted in such a way that the links constituting the ring bus approach from the east and west, and that portals 3 and 4 always face inward toward the center of the ring.

In this experiment, peers are represented as five portals arranged in the shape of a "Y" (fig. 5-10). The stem of the "Y" is defined by three portals that form contact points for the ring bus. The remaining two portals serve both the connections to peripherals and the auxiliary connections to other peers. Reviewing this geometry against the four criterion listed above:

1. For an ensemble with a particle density of 15 per in^2, and an average neighborhood size of 15 particles, the footprint of the peer would be 3.4 × 1.9 in   comparable to that of a slim PDA[10].

2. The relative position of another peer can be expressed as a distance and an angle, both of which can be estimated from a Gradient emanating from the external peer[11].

3. Recoursing to the ergonomics of the situation[12], peers are restricted in their orientation such that the line defined by the three stem portals is approximately perpendicular to the nearest table edge [13]. This insures each ring bus portal an

---

10. and average density of 15 particles / in^2 together with an average neighborhood size of 15 particles yields a communication radius of 0.56 in. Referring to the dimensions of figure 5-10 (normalized to units of communication radii), the peer's dimensions come to 3.27 in. × 1.69 in. with no allowance for borders.

unobstructed path to the corresponding portal of the closest neighboring peer on either side.

4. The restricted orientation likewise insures that the two auxiliary portals (portals 3 & 4 of fig. 5-10) of every the peer face inward toward the center of the ring.

Each peer's behavior is governed by a separate finite state machine (FSM), which can access the peer's portals individually. The input space to the FSM consists of data from the pfrags that migrate into the portals, and the mirrored posts from the HomePages of the neighboring particles. The FSM output space likewise consists of posts to the pseudo-HomePages of the portals and pfrags launched from the portals. The line through the stem portals divides the peer into two symmetric sides. The FSM uses individual control over portals #3 and #4 to break the symmetry and independently direct the activity on each side of the peer.

Extending the Peer with Buoys. Early implementations of Surface Bus confirmed that straight line connections between the ring bus portals suffice to construct ring busses. However, they also highlighted the need for a finer degree of positional control in the immediate vicinity of the peers, where the congestion from the converging Channels tended to block the two auxiliary portals, making additional networking with the peer problematic.

The solution adopted here employs Buoy pfrags[14] to delineate a series of approach paths in the vicinity of the peer. Prior to establishing external links with its neighboring peers, a newly introduced peer first deploys a set of Buoy pfrags. Buoys are migrating pfrags that take up position relative to the fields radiating from selected

11. The incoming Gradient deposits posts in the pseudo-HomePages of the portals. The distance to the source can be read directly from these posts. The angle can be estimated by comparing the distance in the posts at any two portals. The final value is the weighted sum of these pair-wise estimates, with emphasis given to those portals pairs that are most incidental to the gradient field.

12. This orientation is a frequent consequence of users who sit flush against the table and who position their appliances directly in front of them.

13. Empirical evaluation suggest that the orientation of a peer can vary by as much as 30 degrees.

14. As the name suggests, Buoys pfrags are functionally analogous to the buoys which mark the sea lanes in and around a harbor.

portals (fig. 5-11). The Buoys position themselves in such a way as to direct traffic away from the directional portals (#3, 4), leaving them clear to service auxiliary connections to other peers. Once in position, a Buoy radiates its own MultiGrad field that the pfrags of Channel Operators use to taylor their approach[15].

Peer-to-peer Links. At the next level up, where Buoy deployment is treated as a single step, two peers cooperate to construct a multi-channel link. Initially, each peer radiates a single signature Gradient as a vehicle for signaling to the other. Portals sample distance values from the complementary signature Gradients in order to establish the peer's relative position. ID information embedded in the Gradient posts allow the two peers to assign themselves roles for the sequenced construction of the link; with one peer issuing the "homing" Gradients and the other peer staggering the launch of a Channel Operator from each of its three portals. Fig. 5-12 provides additional detail on this sequencing and illustrates the final result.

Ring Bus Formation. At the next level up, where construction of the link is treated as a single step, multiple peers coordinate to construct multi-hop ring busses. On contact with the particle ensemble, a peer emits a signature Gradient and reacts accordance with the number of other peers that it detects. If no other peers are present, the new peer simple idles. If only one other peer is present, the two peers construct a single connection, corresponding to an open ring. Otherwise, the peer integrates itself into the existing ring structure. This involves reading angle and distance for each existing peer, selecting the two peers that constitute the nearest neighbors on the ring, and coordinating with them to replace their existing link with two new ones. Fig. 5-13 illustrates the ring structure that results from the sample placement of four peers. Note how the Buoys direct the Channel positioning in such a way that two auxiliary portals maintain an unobstructed path toward the center of the ensemble.

---

15. In this experiment, the Buoys initiate fields via MultiGradCenterPosts whose payload contained identifiers for the peer, the portal, the direction (east / west) and the "step" (number of Buoy-hops along the approach). The Channel Operator of chapter 4 are adapted to take advantage of this info. In the process of connecting a portal of one peer (the "source") to the corresponding portal of another peer (the "destination"), the Tracers of the adapted Channel Operator sequentially orient themselves to the Buoy from the source, the Gradient from the destination, the Buoy from the destination and finally the Gradient from the destination.
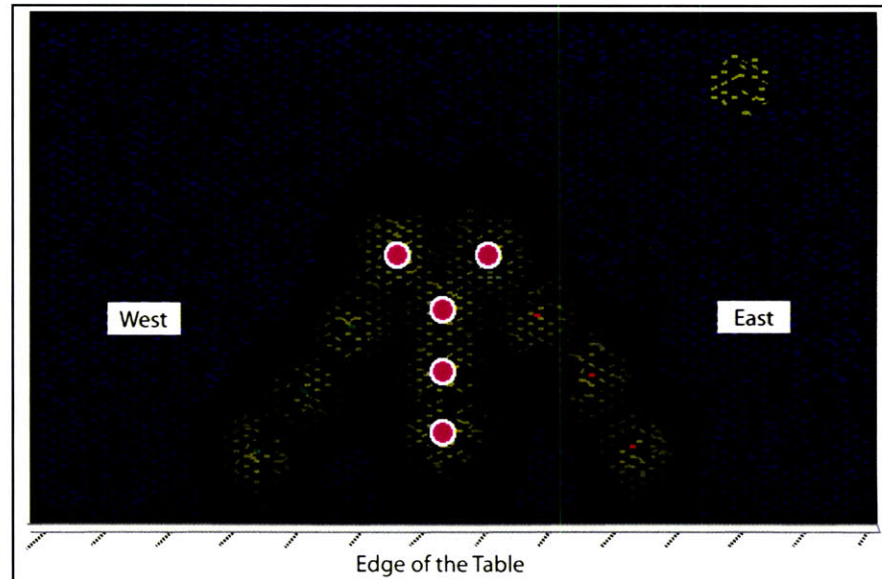
West

East

Edge of the Table

FIGURE 5-11. A Peer and associated Buoys

When a peer makes initial contact with the particle ensemble, the portals all radiate a gradient field via MultiGradCenterPosts. The payloads of these fields contain both the peer-specific ID and a channel (portal) -specific ID. Once the fields have settled, portals $0 \rightarrow 2$ deploy Buoy pfrags, which initially orient themselves to the fields from portal 3 or 4 (in order to break the local symmetry), and then migrate to preferred positions, triangulating off of the fields from selected portals. Once in position, the Buoys radiate their own fields which include the additional ID info for the "path" (east or west) and a "step" (the number of Buoy-hops along that path). Incoming Channels orient themselves to these fields in order to fine tune their approach.

In this figure, Buoys take up staggered positions to define six approaches — three in each direction — each consisting of a single step. Particles containing the westward Buoys are shown in green. Those containing the three eastward Buoys are shown in red. The highlighted particles in the upper right illustrate the size of a single network neighborhood. Each Buoy orients itself to the fields from three portals. For each Buoy, the table below lists the three anchor portals and the Buoy's preferred distance from each anchor (*italics* in units of communication radii).

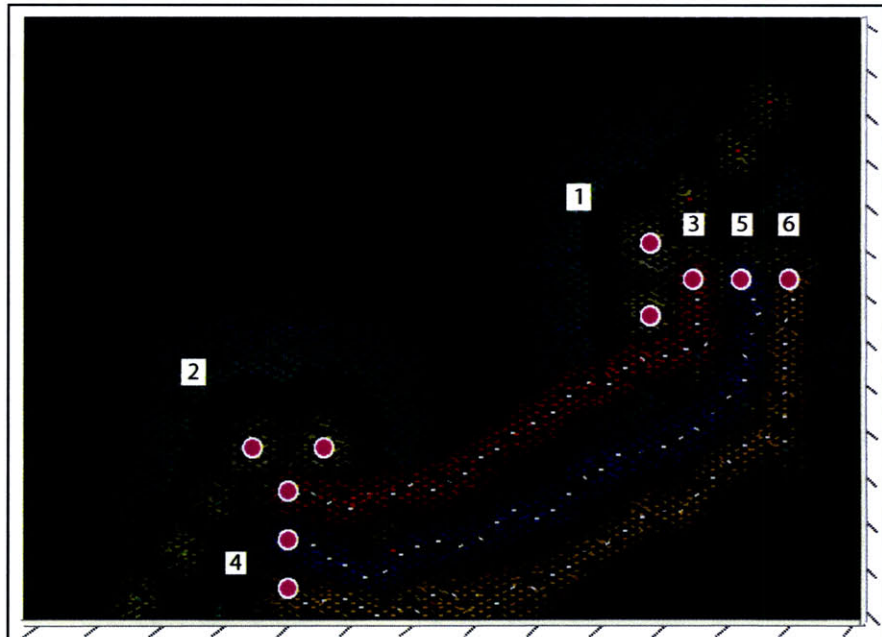| Channel ID number	Anchor Portal 1		Anchor Portal 2		Anchor Portal 3			
					east	Buoy	west	Buoy
0	0	*3.0*	2	*5.0*	3	*2.34*	4	*2.34*
1	1	*5.0*	2	*5.39*	3	*5.17*	4	*5.17*
2	2	*7.0*	0	*8.06*	3	*8.0*	4	*8.0*

FIGURE 5-12. Peer-to-peer Link

A link connecting two peers is constructed from three Channel Operators that grow connections between the corresponding portals of the two peers (See Channel Operator on page 87.). Once in place, Channels grow mini-Gradient fields that act as a sheath and repulse neighboring Channels, thus inhibiting overlap between the Channels.

Link construction is sequenced as follows (numbers matched to those in figure).

1, 2: Peers make initial contact with the particle ensemble, deploy their Buoys and radiate a signature Gradient from their respective portal #0. By sampling the Gradient distance at multiple portals, a peer estimates the angle of the originating peer and uses this angle to decide which side to build the link on. Peers also compare ID info embedded in the posts of each other's Gradients in order to assign themselves the role of "transmitter" or "receiver". In this figure, peer #1 is the "transmitter" and peer #2 is the "receiver".

3: Transmitter peer initiates a Channel Operator that connects the two portal #0's.

4: Receiver peer radiates "homing" Gradients from its portal #1 and #2 (concurrent with step #3)

5: On arrival of the homing Gradient for portal #1, the Transmitter initiates a Channel Operator to connect the two portal #1's

6: After a selectable delay, the Transmitter initiates a Channel Operator to connect the two portal #2's
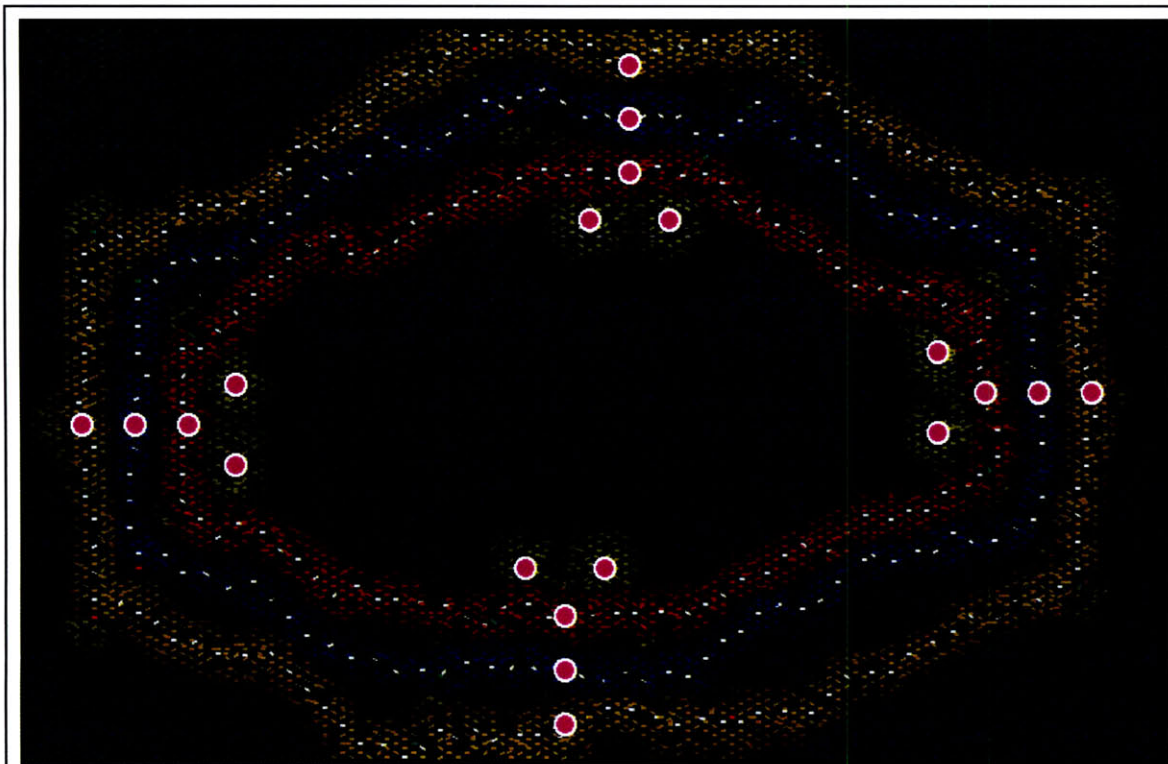
FIGURE 5-13. Ring Bus formation

Four peers construct a multi-hop network in the shape of a ring. Signature Gradients radiated by each peer enable the peers to establish their relative position, select their nearest neighbor on each of two sides, and construct the multi-channel links.

In this figure, the placement supports a closed ring. For the case when a peer finds no neighbor on one of its sides (east or west), only one link is constructed and the ring is open.

Concomitant Processing. A natural advantage of message passing on a paintable is the option for concomitant processing — processing that is executed on the message data as a side-effect of the transmission. This functionality can assume many forms. A simple proof-of-principle embodiment is that of a plug-in replacement for the 2-port communication-only Channel Operator. In this 2-port Process-Channel Operator, data is likewise streamed through the end points, enveloped in Carrier pfrags as payload. However the intervening particles are overlaid with a distributed procedure that operates on the propagating data in a manner reminiscent of data flow architectures.

Given an unobstructed straight-line path between the two portals, a simple implementation of a Process-Channel Operator can be built around the Coordinate Operator of chapter 4 (pp. 91):

- Use the Coordinate Operator to establish a local 2D coordinate system between the two portals.

- Map the selected procedure to a 2D planar graph, overlay a graph-relative coordinate system, and express the nodes of the graph as (perhaps multiple) mobile pfrags with assigned 2D coordinates (fig. at right)

- Diffuse the pfrags into the particle ensemble where they orient themselves to the coordinate system established by the Coordinate Operator.
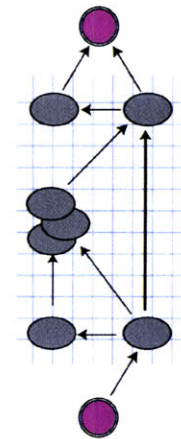
In the ring topology of Surface Bus, the particles in center of the ring constitute a shared processing resource. Any two peers can engage this resource by connecting through a Process-Channel Operator. In order to avoid spatial overlap, pairs of peers must first confer over the ring bus with the other peers in order to arbitrate their access. On completion, the resource can be released by signaling a termination to the Process-Channel Operator.

Discussion. This presentation of this section is illustrative on three distinct levels. At the most abstract level, this application is an emblematic of those applications that structure the processing based on an estimate of the local geometry. Encompassed as a subclass are those applications that first use patterning techniques to erect a scaffold and then express a distributed procedure as a collection of mobile

fragments that are selective to positions on that scaffold. With its reliance on accurate distance estimates, this class of applications is explicitly dependent on the average number of particles falling within a communication radius[16]. By contrast, in the previous two applications (Steaming Audio and Holistic Data Storage), the core functionality was diffusive scattering and did not require precision placement[17]. Likewise in the next application (Image Segmentation), the geometry is implicit in the input image and need not be explicitly estimated

The crucial observation of Surface Bus is that *the growing expertise in the generation of complex patterns can be used to direct a similarly complex flow of control and data within a distributed process*. With this in mind, the objective of Surface Bus was not to demonstrate virtuosity at emergent patterning, rather to illustrate how even simple geometry estimation can underlie a broadly useful functionality.

At the second level of abstraction, Surface Bus addresses the general problem of dynamically partitioning a particle ensemble into distinct functional regions. In this context, the *paintable* is an early exemplar of a "computing substrate whose computational architecture emerges from the information that passes through it".[18] In Surface Bus, the emergent architecture is that of a dynamic communication network connecting multiple clients who in turn coordinate to share a pool of reconfigurable processing. The utility of this architecture extends to the general instance of any particle ensemble contained in an arbitrary surface or volume, and a set of clients that interface to the ensemble along its periphery[19].

---

16. The accuracy of gradient-based distance estimates, as a function of both density and positional randomness, has been treated analytically [40]. Resolution in the distance estimate has been shown to be asymptotic in the size of the communication neighborhood, with 15 as a commonly accepted number for the point of diminishing returns.

17. Even the call-back functionality was robust against noisy distance estimates in the CallBackGradients.

18. This paraphrases a particularly insightful remark by H. Shrikumar and Neil Gershenfeld. [Personal communication]

19. Consider an instance of a particle ensemble encased in an arbitrary 2D slab — comparable in size to a white board — and clients (peers) that interface via tethered, male multi-pin connectors.

Finally, at the lowest level of abstraction, we consider Surface Bus as a distinct, well-defined application in which thousands of micros are embedded into a tabletop and communicate wirelessly with multiple devices scattered about the tabletop's periphery. In this context, Surface Bus spotlights looming issues in both product design and system's engineering.

The infusion of fine grain computing into environmental mainstays such as furniture opens up a fresh design space with heightened potential — and peril. By ingraining gigaFLOPs of compute capacity into some of man s oldest domestic artifacts, designers run the risk of promulgating unnerving complexity deeper into the personal sphere. Surface Bus illustrates a representative solution based on past practice[20]. Here, the conventions and gestures commonly associated with a table provide a ready vocabulary for management of a communication network. For example, a peer s membership in the network is signaled gesturally by placing the artifact on the table. Similarly, removal of the artifact is a natural signal for its exclusion from the network[21]. The size of a peer is suggestive of the amount band-width the corresponding multi-hop node can support. And latency likewise maps to physical distance, with a cluster of physically proximal peers enjoying low latency interconnects. The size of the table is an immediate visual queue for the raw processing capacity available from the table. And crucially, in an instance when that

---

20. Personal computing has always been a challenge for product design. The design mantras of observability, consistency, robustness, and conceptual transparency are perennially at odds with the nature of computing hardware, whose operation is inherently complex, opaque, seemingly arbitrary and fatally sensitive to errant behavior on the part of the user. Designers, applying a mature set of guidelines, seek to bridge this gap on a case-by-case basis [42]. Early success with VisiCalc validated the strategy of casting the computer as a chameleon — hiding the computing behind an image or form whose commonly understood affordances proffer a rich, intuitive set of guidelines to aid the user. VisiCalc used the then-novel interfaces of a mouse and a bitmapped display to project the appearance of a spreadsheet that could automatically adjust the entire document to reflect changes made to any single entry. Widespread familiarity with traditional hard-copy spreadsheets provided a detailed mental model to bootstrap the use of the programmable extension. As ever increasing compute power outstripped the representational capacity of the standard interfaces, researchers began exploring the use of "Tangible Interfaces" — ordinary artifacts such as bottles, clocks, fans and even projected weather patterns whose commonly understood conventions are extended and mapped to the behavior of the computing environment [48] [49] [21]

21. The gesture of placing an object on the table is an age-old signal for the object s release to a larger group. Consider the instance when an architect "puts the plans on the table".

---

capacity eclipses the compute capacity of the individual peers, the computational limitation on any single peer is decoupled from the peer s internals. In other words, the conversation between a key chain and a fountain pen can be (almost) as computationally rich as the conversation between two laptops.

On the systems engineering side, the noteworthy topics are:

* alternate approaches for functionally partitioning the particle ensemble.

* possible refinements and extensions to this particular approach

* basic engineering limitation for Surface Bus as described.

The overall goal of functionally partitioning the particle ensemble can be served by a variety of techniques. These techniques are differentiated by the degree to which the partitioning is centrally directed. At the extreme of centralization, a Coordinate Operator first establishes a spatial ordering in which each particle is uniquely addressable. A single agency[22] then dictates the layout and construction of the links between the peers, using a program that searches for a global optimum[23]. At the opposite extreme of decentralization, the network topology is an emergent property of the strictly local interactions among mobile pfrags that mimic the trail-laying behavior of ants. Simulated pheromones couple the optimality of the overall link topology to the frequency with which the constituent paths are traversed. The resulting network topology is both dynamic and usage dependent. Similar techniques for adaptive routing have been proposed for load balancing in telecom networks [44].

The Surface Bus approach of this section falls in the middle ground. The spatial regions for communications and processing are segregated a priori into regions whose performance can be analized separately. However this segregation is expressed as an emergent behavior. The resulting system represents a trade-off between centralized control (which is antithetical to this research) and unbounded emergence (whose performance can be difficult to bound analytically).

Surface Bus, as presented here, can be improved in at least two ways:

---

22. Possibly a permanently attached peer or single connection to a larger computing resource.
23. Similar to the programs used for auto-layout of PC boards and IC logic circuits.

- Use a more sophisticated composition of fields to confine the communication ring to the outer most periphery of the table. This would maximize the number of particles that would be free for use a reconfigurable processing resource.

- Extend the Coordinate Operator so that the coordinate system between any two points can be conformally warped, supporting a more efficient use of the particles in the table center[24].

These refinements not withstanding, the core engineering limitations are latency and throughput. In a fine-grain multi-hop network like Surface Bus, transmission latency is a linear function of the path length. The shorter the link, the smaller the latency. For throughput, the fundamental limit is the bandwidth of the inter-particle communication. This limit can be somewhat relaxed through use of links built on multiple channels. But available surface area of the peers will typically confine this gain to a single digit linear scale factor.

---

24. Consider the case where the straight-line path between two portals is tightly confined, while a curved path would incorporate many more particles.

Image Segmentation

The golden rule in massively parallel computing is to match the topology of the machine to the natural topology of the problem. In this section, we use image segmentation as an example of a task that is well matched to a common topology of a paintable — namely that of a 2D plane.
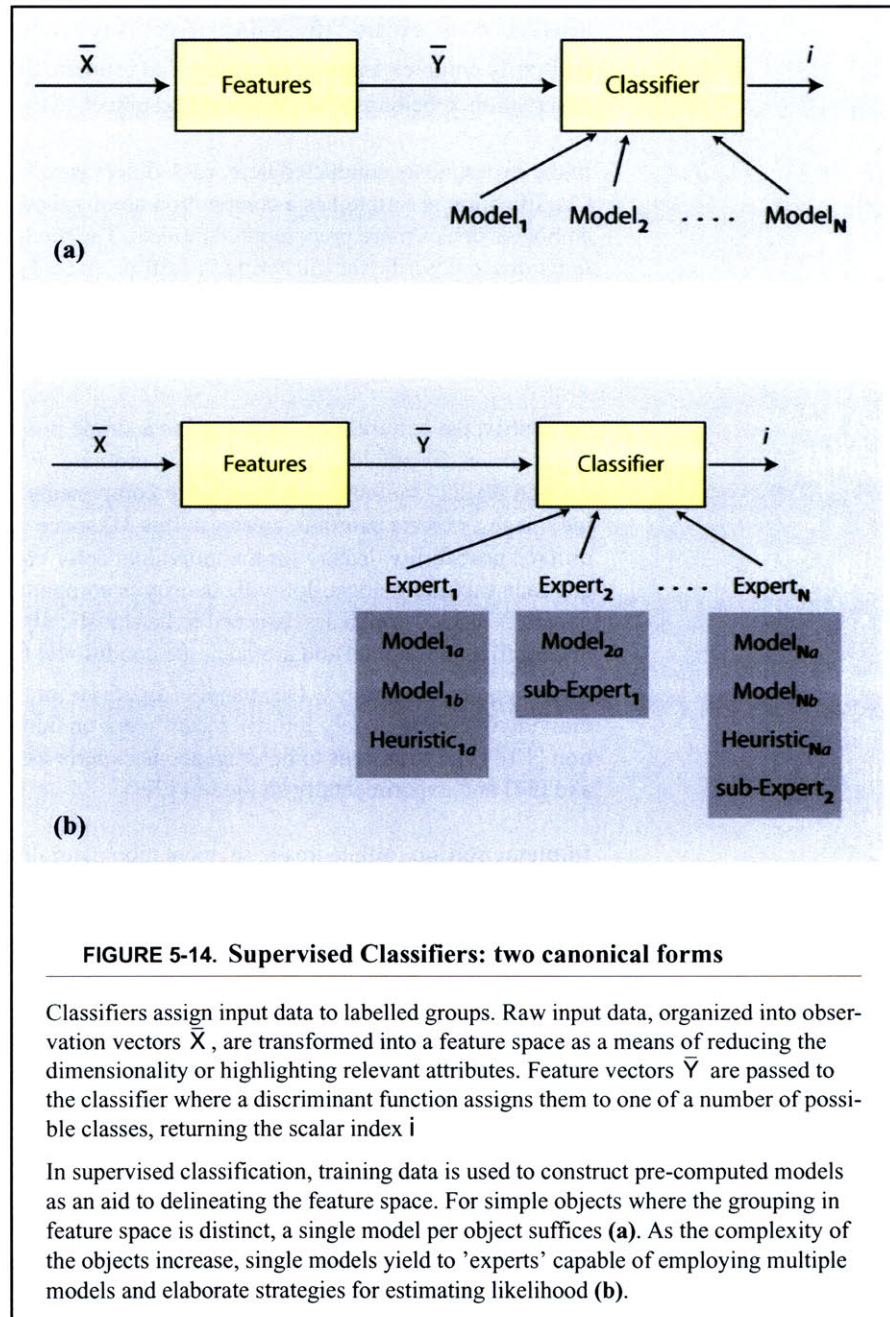
The structure of this section follows that of the previous three. A working introduction to the theory — in this case supervised classification via experts — is followed by the details of an implementation on a *paintable*. The algorithmic refinement of the segmenter is kept purposefully minimal to keep focus on the basics. Experiments illustrate the segmentation behavior on a natural image with four distinct color groups. A review discussion highlights crucial attributes.

Segmentation. In image segmentation, individual pixels are assigned to a region corresponding to an object. Here the term 'object' can range in complexity and semantic depth from simple surface discolorations to composite devices consisting of many moving parts. Segmentation techniques likewise range in sophistication from simple foreground / background separation via pixel thresholding, to human assisted paint programs for colorizing monochrome film stock.

Image segmentation is an instance of the general problem of classification. Fig 5-14 illustrates two important canonical forms. Novel, unclassified input data is represented as an observation vector $\bar{X}$. Observations are transformed into a feature vector $\bar{Y}$, as a means of reducing the dimensionality and/or highlighting salient characteristics. Feature vectors are passed to a classifier where a discriminant function assigns $\bar{X}$ to one of a number of possible output classes, and reports the selection as a scalar i

In the supervised classification techniques of interest here, the number of objects is known *a priori* and training data is analyzed offline to construct a separate model for each object. Given a novel feature vector, the classifier uses these models to estimate the likelihood that the input observation belongs to the associated object[25]. For simple objects and a suitable feature space, the grouping in feature space is explicit and the single-model-per-object configuration is sufficient to support classification (fig. 5-14a). As the complexity of the objects increases, the

---

25. For example, a model can be a statistical description of feature space, in which case the classifier uses hypothesis testing as a discriminant function.

FIGURE 5-14. Supervised Classifiers: two canonical forms

Classifiers assign input data to labelled groups. Raw input data, organized into observation vectors $\overline{X}$, are transformed into a feature space as a means of reducing the dimensionality or highlighting relevant attributes. Feature vectors $\overline{Y}$ are passed to the classifier where a discriminant function assigns them to one of a number of possible classes, returning the scalar index i

In supervised classification, training data is used to construct pre-computed models as an aid to delineating the feature space. For simple objects where the grouping in feature space is distinct, a single model per object suffices **(a)**. As the complexity of the objects increase, single models yield to 'experts' capable of employing multiple models and elaborate strategies for estimating likelihood **(b)**.

representational capacity of the models is outpaced, and the models are replaced by heterogeneous 'experts' (fig. 5-14b). Experts typically employ multiple models and arbitrarily complex analysis strategies[26] to estimate the likelihood that the input observation $\bar{x}$ belongs to their associated object.

In the experiments conducted here, each object is represented by a single expert. Classification is scripted as a competition among these experts. Each expert embodies one or more precomputed models. The models use parameterized density functions to describe the clustering in feature space. For each incoming feature vector $\bar{Y}$, an expert employs each of its models to separately estimate the likelihood of $\bar{Y}$. The greatest likelihood is returned as the expert's input to the classifier, which in turn uses a simple max() as a global discriminant function.

For clarity, the feature space is limited to a single image attribute — color. Input observations are single pixels and their coordinates in the 3D feature space is defined directly by the values their color components. For a restricted class of simple images, objects naturally cluster in this 3D space (example; fig. 5-15). The conditional probability density for the individual color components is estimated using gaussian mixture models. Joint 3D density is computed as the product of the component densities, which are assumed to be statistically independent. Given an unclassified input pixel and a model, the conditional likelihood is computed via MAP hypothesis testing [27] (see panel of fig. 5-16 for detail). This definition for the individual models closely follows recent work on human-assisted image segmentation [10][11]. Extensions to heterogeneous experts were treated formally in [32] and [54] and experimentally studied in [9].

Implementation. While image segmentation naturally maps to processing on a 2D particle ensemble, implementation presents three challenges:

· delivering the input image data in parallel

· distributing the processing

· compensating for the irregular particle lattice.

---

26. Including the hierarchical application of other experts.

27. Maximum a posteriori (MAP) hypothesis testing: compute the conditional probability of a novel input feature vector given that the input belonged to a given object.

FIGURE 5-15. A natural image that is easy to segment

With minimal exceptions, a feature space built on color alone is sufficient to segment the image into four distinct objects: sand, water, sky and tree. Ambiguity occurs only where the waves break on the beach. Here, the saturated pixel values are indistinguishable from those of the clouds.

The first problem is the distribution of the image pixels among the particles. The pixel positioning should be image-topic in that pixels that are spatially adjacent in the image should occupy particles that are themselves proximal. The presentation of the previous sections suggest the following approach:

- use pfrags diffusing from two anchor points to construct a coordinate system.
- envelop the pixels in Carrier pfrags designed to transport the pixel to the correct position on the coordinate system.
- diffuse the Carriers into the particle ensemble through multiple portals.

The drawbacks of this approach are the bandwidth bottlenecks at the portals, and the large variance in latency between the moments when the first and last pixels arrive at their position.

The basic component of a mixture model is a gaussian described by three parameters: the mean $\bar{x}$, the variance σ^2, and a weight $\mathbf{w}$.

$$G(x, \bar{x}, \sigma, w) = w \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x - \bar{x})^2}{2\sigma^2}}$$
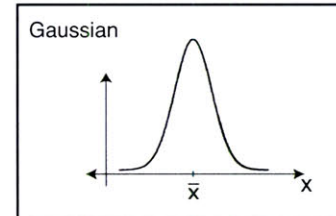
Gaussian

Expectation maximization is used to fit multiple gaussians to the histogram of training data. The resulting mixture model can be used to estimate the likelihood of a new scalar observation $\mathbf{x}$

$$P_c(x) = \sum_{i=1}^{N} G(x, \bar{x}_i, \sigma_i, w_i)$$

Histogram/ PDF estimate

Pixel value
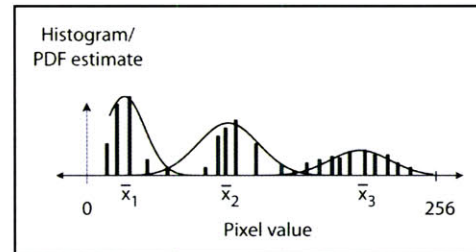
Given an unclassified M-dimensional feature vector $\bar{\mathbf{y}}$, the likelihood of $\bar{\mathbf{y}}$ is estimated by the product of the likelihoods of the M scalar components $\mathbf{y_k}$

$$P_Y(\bar{y}) = \prod_{k=1}^{M} P_{C_k}(y_k)$$

FIGURE 5-16. Mixture Models for Feature Discrimination

An attractive alternative is to employ simulated photo sensor particles — pseudo-particles whose size is comparable with the generic processing particles and whose networking behavior mimics that of the particles. Sensor particles are pseudo-randomly distributed among the processor particles, with a density consistent with the desired average sampling resolution.

Image capture would proceed in parallel using the hypothetical apparatus of fig. 5-17. A suitably bandlimited image would be projected onto the plane containing the particle mixture. Sensors should sample the local intensity, encode the pixels as MultiGradCenterPosts and post them to their pseudo-HomePages. MultiGrad pfrags in the neighboring processor particles propagate this data outward from the sensor in the form of mini-Gradient fields, which carry the pixel data as payload. The radius of these fields is selected such that the HomePages of processing particles will contain between 5 and 15 such posts, each containing both pixel data and an estimate of the distance to the originating sensor.
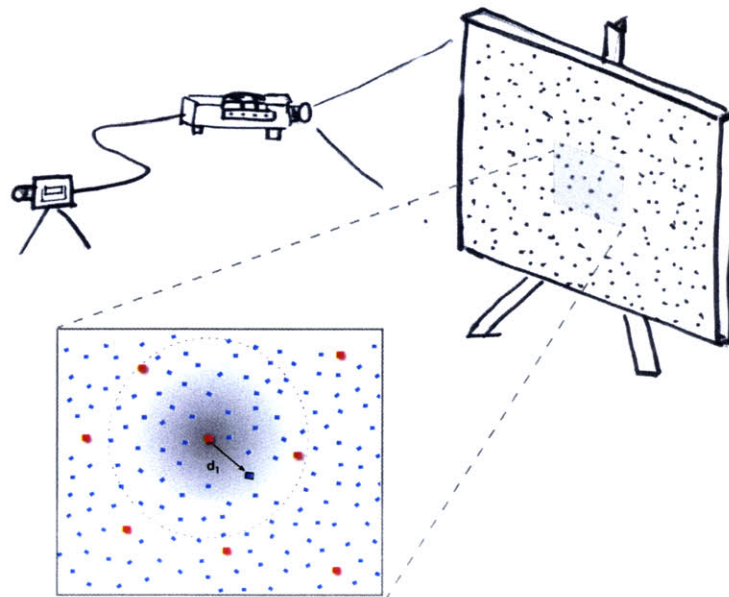

FIGURE 5-17. Proposed apparatus for parallel image sampling

A projector positions an image on a plane containing a mixture of photo sensor (red) and generic processor particles (blue). Pixel values radiate locally from each sensor, carried as payload in a MultiGrad gradient field. The range of the fields are specified such that every processing particle contains pixels from multiple sensors.

Segmentation is scripted as a competition among "experts", each estimating the MAP likelihood that a pixel belongs to a particular object. Each expert is embodied as a pfrag which enters, multiplies and diffuses to position a copy of itself in every particle neighborhood. Experts report the likelihood in HomePage posts. The discriminant function is a simple max().

Experimental Results. Segmentation performance was illustrated qualitatively on the simulator. Fig. 5-18 shows the test configuration. 1800 sensor particles are pseudo-randomly scattered among 5000 generic processor particles. Training data was drawn from the beach scene in fig. 5-15 to create four experts, each selective for one of the four basic object groups in the image: *sand*, *water*, *sky*, and *tree*. Each expert used the training data to synthesize one or more mixture models for the probability density in RGB feature space.
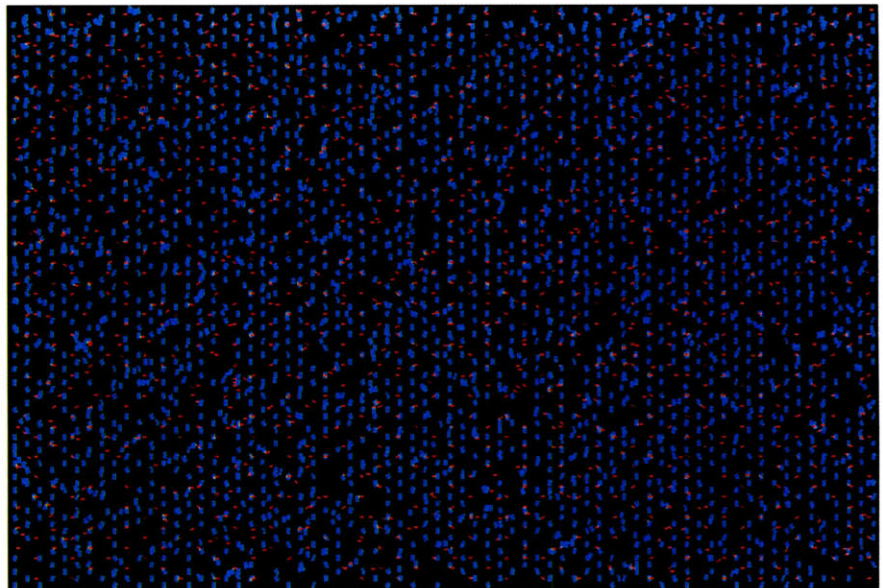

FIGURE 5-18. Image Segmentation: Initial state

1800 sensor particles (red icons) are pseudo-randomly scattered among 5000 generic processor particles (blue icons). Both particle types have identical communication range. Processing particles have, on average, 14 neighbors and are in contact with 5 sensors.

Experts propagate copies of themselves to every particle in the ensemble. Once in place, experts examine the pixel data in the particle HomePage, select the pixel from the closest sensor, estimate the likelihood that the pixel belongs to the object, and posts this likelihood as a score on the HomePage (fig. 5-19). For each particle, the viewer scans the HomePage, and color codes the particle icon to reflect the expert with the highest score.
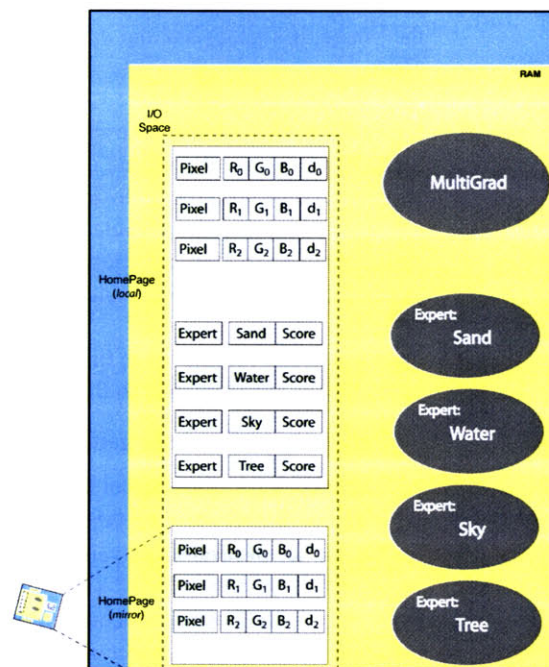
FIGURE 5-19. Image Segmentation: pfrags and data

In the steady state, all processor particles contain 5 pfrags: a MultiGrad and a copy of the four object expert pfrags. MultiGrad posts contain pixel data radiating outward from the nearby sensors. Experts pick the pixel from the closest sensor, and apply their internalized models to estimate the MAP likelihood that the pixel belongs to the associated object. Note that, although this test examines the RGB feature space only, the pixel data in the MultiGrad HomePage post contain sufficient data to include texture as a feature as well.

Fig. 5-20 illustrates the operation and shows the end result. The original beach image is projected onto the heterogeneous particle ensemble where the sensors sample the intensity. In fig. 5-20*a*, the pixel data has been distributed and the viewer color codes each particle with the pixel value from the nearest sensor[28]. In figs. 5-20*b* through 5-20*e*, the experts are diffused sequentially into the ensemble. With the arrival of each expert, the segmentation monotonically improves. In fig. 5-20*b*, the only expert present is the sky expert, which wins every competition by default. In fig. 5-20*c*, the tree expert dominates over the tree and water portions of the image, In figs. 5-20*d* and *e*, the remaining two experts arrive and stake their claim.
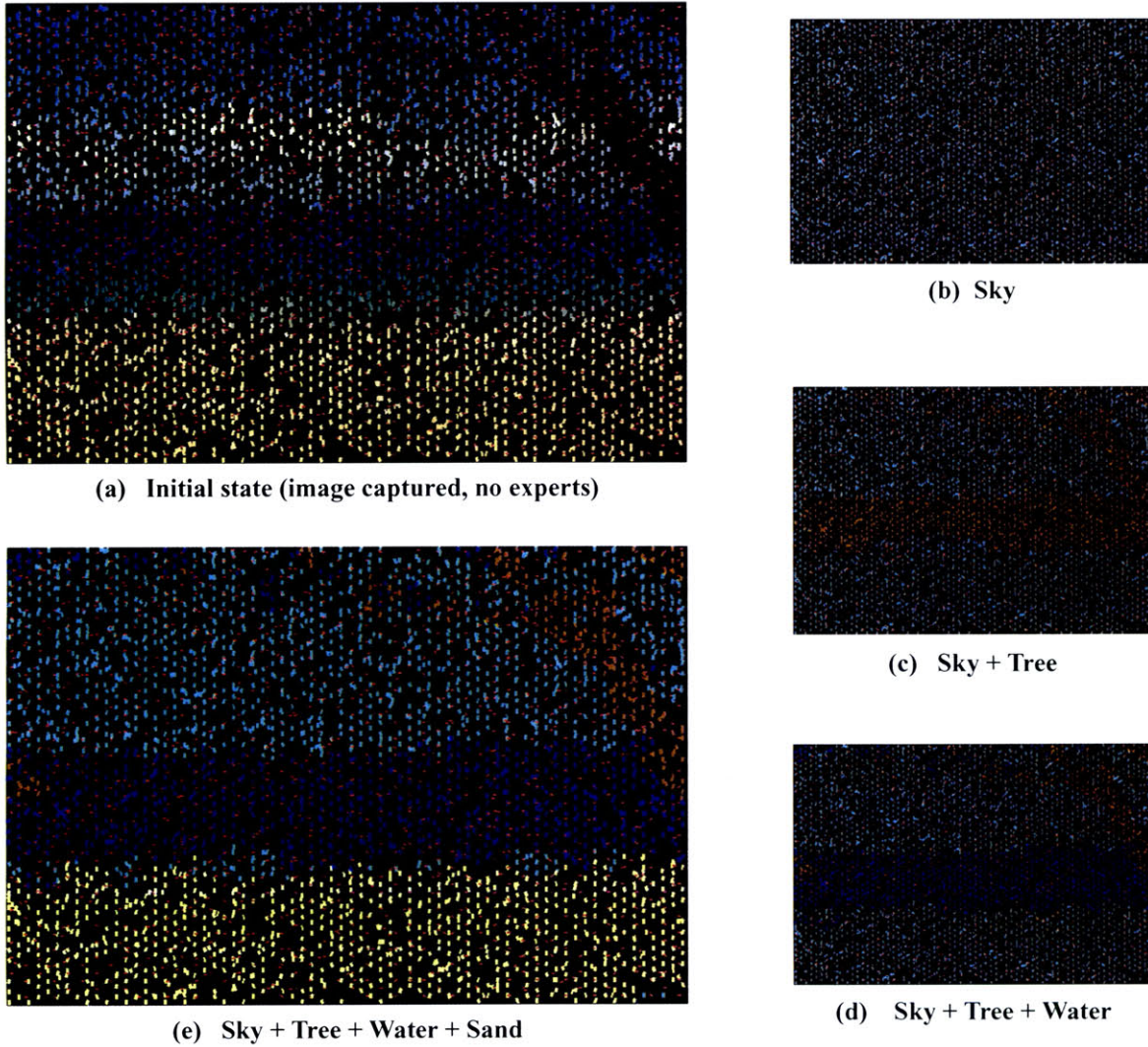
Discussion. The approach described here is purposefully simple and would benefit markedly from even moderate engineering refinement. Classification robustness would improve with the inclusion of more features. For example, from the data already available in the particle HomePages, experts could estimate image texture. And with support from additional pfrag types, the feature space could be further expanded to include motion and position.

Yet even in this simple form, this application underscores a number of crucial properties that are fundamental to this genre of computing:

- Raw price/performance gain for a distributed architecture.
- The advantage of proximal computing —
  positioning the computing near the sensing devices.
- The utility of pfrags as a vehicle for knowledge representation.

Fig. 5-21 coarsely estimates computing performance assuming some representative values for clock speed, power, and number of processor particles. The dearth of detailed designs for the various particle subsystems limit the accuracy of these figures, admitting errors in the exponents on the order of ± 2. However concern over the accuracy of the estimates masks the larger point that the compute capacity of this architecture is simply huge, vastly outweighing foreseeable generational progress in contemporary architectures.

---

28. As read from the distance value in the HomePage posts.

(a) Initial state (image captured, no experts)

(b) Sky

(c) Sky + Tree

(d) Sky + Tree + Water

(e) Sky + Tree + Water + Sand

FIGURE 5-20. Image Segmentation: region assignment among experts		Sky	
		Tree	
(a) The particles are color coded to show pixel from nearest sensor.		Water	
(b) —(d) Ensemble state after sequential introduction of experts for sky, tree, and water (respectively).		Sand	
(e) Final assignment after all four experts are present.			

Equally important is that image segmentation as a problem domain makes effective use of this capacity. By varying the relative ratio of sensor to processor particles, designers can deliver an average <u>per pixel compute capacity</u> in the range of 50 to 500 MIP's[29]

System Specs:

No. of particles	4.8×10^5
Clock rate	50×10^6 Hz
Ensemble dimensions	2m x 1.5m
No. particles per neighborhood (ave)	15
No. sensors per neighborhood (ave)	5
cost per processing particle	$0.03
power dissipation per particle	1×10^{-2} W

Total Power Consumption: 4.8×10^3 W
(No. of particles) x (power dissipation per particle)

Areal Power Density: 1.6×10^{-1} W/cm
(power dissipation per particle) / (areal size of ensemble)

Total Compute Capacity: 2.4×10^{13} inst/sec
(No. of particles) x (particle clock rate)

Total Cost: $14,400
(No. of particles) x (cost per processing particle)

FIGURE 5-21. Image Segmentation:
     Sample specs and estimated performance

Performance figures for an image segmentation system are derived from rudimentary estimates for the particles specs and the layout of the 2D ensemble. Lack of working hardware (particles) frustrates detailed calculations and forced the use of conservative figures as a stand-in. Nevertheless, the final result of US$14,000 for 24 teraFLOPS of utilized compute capacity could sustain small digit errors in exponents and still be compelling

Approaches to image understanding often treat pixel-based region assignment as an early stage of visual cognition. From this perspective, an implementation based on autonomous pfrags interacting in an open competition is intriguing. In his now pervasive theory of cognition and perception, Minsky[39] modeled intelligence as aggregate behavior of a heterogeneous society of numerous simple agencies acting in concert to solve difficult problems. Reduction to practice has always posed two challenges: a representation of knowledge based on distinct agencies, and a set of organizing principles to govern their interaction. Proposed solutions to these two questions must jointly address problems of adaptation and scaling — both basic necessities for growing the society in a dynamic environment.

In the classification technique advanced here, knowledge is represented by autonomous experts programmed with flexible strategies for migration, grouping and interaction. Organization and regulation is based on an open competition structured around the fundamentals of the problem domain[30]. This approach supports several important properties.

- Experts can be trained separately by different people at different times to be selective for different objects.

- Novel experts can be streamed into the ensemble at any time and from any point of contact. They diffuse throughout the ensemble, and position themselves in accordance with competitive pressure.

- Finally, the introduction of a new expert effectively makes the ensemble selective for a new object. And with each new expert, the overall classification performance improves monotonically[31].

---

29. Varying between 1 and 10 the average number of processor particles proximal to each sensor.

30. In this case, the physics of image formation and the statistical relationships between sampled intensity data.

31. This is only true in the statistical sense. Given additional experts selective for objects not in the scene, the likelihood of misclassification actually increases.

Discussion

This chapter focused on two questions: "*can this programming model do anything useful?*" and "*if so, can it do any of those things particularly well?*" To the first question regarding practical utility, the answer is an unambiguous "yes". Even discounting the implementation details from all four examples, this chapter has demonstrated the programming model's basic support for storage, communication and signal processing — building blocks which are necessary and sufficient for a wide variety of important applications. And while the limits of the application space remain unexplored, the results from this chapter suggest that candidate applications can be as novel as the architecture itself (e.g.. Holistic Data Storage).

To the second question regarding optimality, the answer is more subtle. On first inspection, the picture is not promising. In their nominal configurations, all the examples of this chapter can be more efficiently implemented using conventional designs. Both storage applications could be efficiently realized with centralized address controllers directing access to monolithic memory and engaging simple signal processing as needed. Likewise, the Surface Bus application offers no compelling advantage over near-field RF coupled with pipelined processing on the buffered messages. And while a 24 TeraFlop machine would be nice, their scarcity has not impeded the proliferation of desktop image segmentation tools.

So when does process self-assembly on a *paintable* become the optimal choice? And is it ever the only practical choice? The answers hinge on two observations:

· Complexity, in one form or another, determines the scaling limits of engineered systems.
· Self-organization is a powerful tool for managing complexity.

Taken together, these two items suggest that the cross-over point where process self-assembly becomes efficient is that point where rising complexity overwhelms conventional techniques.

As an illustration, consider the operation of Holistic Data Storage (HDS). Here, the task is to transform the input data into a hierarchical representation and to position the transform samples "holistically" within the memory. In the example of this chapter, both the transformation and the positioning were byproducts of the interaction among migrating pfrags. A functionally equivalent system could be constructed from monolithic memory driven by a memory controller consisting of dedicated logic for address generation and pipelined signal processing for signal transformation (fig.5-22a).

140

In this application, operational complexity can be regarded as a semi-continuous space defined by several axes:

- number of input sources
- variability in these sources (both in instantaneous number and in bandwidth)
- size of the memory (i.e. address space)
- variability in the memory topology (both intentional and unintentional)
- statistical likelihood of fault (or conversely, the minimum required reliability)

Associated with each point in this complexity space is a cost for constructing and maintaining the system — component purchase, manufacture, test, servicing, support, and liability insurance.

Fig. 5-22 portrays the memory controller at two extremal points in operational complexity space. In the simple configuration (fig 5-22b), statically configured memory is storing data from a single source streaming at a fixed bandwidth. Transform hardware is optimized for the tightly bounded operating range. Constraints on input format together with the fixed memory space allow the address generation logic to deterministically compute the addressing that distributes the data "most holistically". In this low complexity regime, the dedicated memory controller enjoys a dominant cost advantage over the *paintable*.

With increasing operational complexity, the situation changes. In the extreme of fig. 5-22c, the topology of memory has become time varying. The number of input sources varies randomly with time, as do their individual duty cycles[32]. Stringent reliability criteria now necessitate both design for the worst case load and additional redundancy for tolerance to fault. Variance in the input load and memory capacity combine to make it impractical to design an address generator that deterministically computes addresses that are always "holistically" optimal. With rising operational complexity, we ultimately arrive at a point where the diffusion model of address generation becomes attractive. At this point, it is the redundant array of costly memory controllers that is emulating the behavior of a *paintable*, rather than the other way around.

---

32. Examples of inputs with variable duty cycles: microphones that only go on when people speak, cameras that only go on when objects move, mechanical strain sensors that only transmit when the readings depart some allowed nominal range
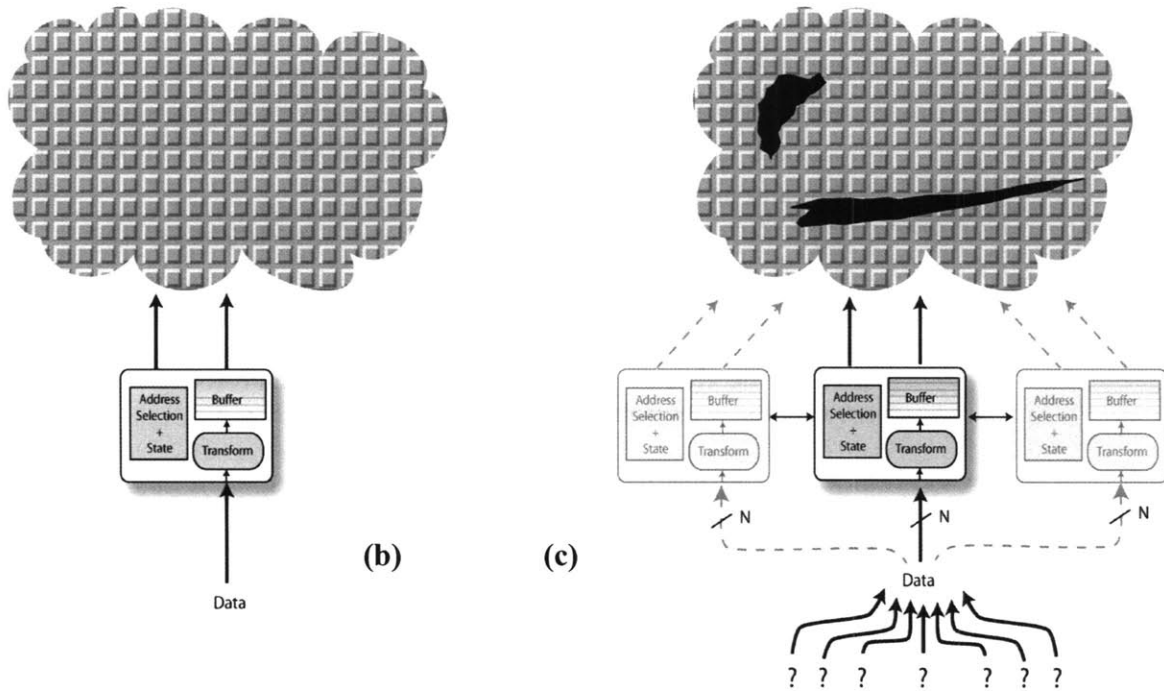
(b)   (c)

Data

(a)

FIGURE 5-22. Conventional memory controller design at opposing extremes of operational complexity

In the conventional equivalent of HDS, a memory controller transforms incoming sequential data into a hierarchical representation and directs the placement of the transform samples into external memory. The utility of the this approach depends on the complexity of the operating environment.

(a) *Functional illustration of a single controller unit.* The transform block generates a hierarchical representation of the incoming data, buffers the hierarchical data for output and passes storage requirement data on to the address generator. Dedicated address generation logic arranges the data holistically in static memory.

(b) *Simple configuration*: Single input source streaming with constant bandwidth. Static memory configuration. Modest reliability requirements.

(c) *Configuration at a complexity threshold*: Multiple input sources, each with highly variable bandwidth. Memory with time varying topology (possibly due to ongoing material failure). Component redundancy necessitated by stringent reliability requirements.

The questions surrounding practical applications of Holistic Data Storage on a *paintable* can be reduced to this: "Is there a point in operational complexity space, corresponding to a commercially viable application, where HDS on a paintable exhibits a dominant cost advantage?". Definitive answers must wait for performance analysis on real systems. But a good starting point would be extensions to existing applications that are operating near their complexity limits[33].

While the above details are unique to the HDS application, this exposition underscores a dynamic that is more universal, namely *as systems scale upward in complexity, designs based on centralized computing hit their limit. Often, these limits yield to an implementation based on finely distributed computing — thus enabling another round of scaling.*

In Surface Bus, the scaling features of interest are the number devices on the table, their physical size and available power. The conventional node count limitation for near-field RF pico-nets is obviated by the expandable ring bus of the paintable implementation. Likewise the 5 cm communication path between a peer and the nearest *paintable* particles lowers the minimum power requirements for the peers, thus admitting smaller devices traditionally devoid of serviceable battery storage (pens, key chains, shot glasses, small books).

In the image segmentation application, the considerations are straightforward. The important scaling features are the sampling density, the number of experts and the throughput. These trace back to a single system attribute; the average compute capacity per pixel. In conventional designs, the upper bound on performance is set by the compute capacity of the processor divided by the number of pixels. In the

---

33. Consider the case where HDS is employed as a substitute for the "black boxes" of an airplane. In the HDS alternative, ensembles of particles are embedded into the airframe and wings. The conventional input space (e.g.. voice data from the flight deck) is expanded to include mechanical and chemical sensors that have likewise been embedded through out the aircraft in physical proximity to portions of the particle ensemble. Bandwidth from this sensor ensemble varies from a slow trickle under normal conditions to an avalanche of data in a moment of catastrophic failure.

In the tragic instance of the plane crashing in an environment that naturally hinders the recovery, initially recovered parts (e.g.. a fragment of a wing) will yield a muffled recording of audio from the flight deck and summary readings from the sensors. With each additional part recovered, the record of events would come into sharper relief.

paintable implementation, both the number of pixels and the total compute capacity can be selected by varying the number and density of both particle types. This strategy is ultimately bounded by the mechanical limitations of weight, packaging density and power dissipation.

Summary

This chapter was devoted to exploring the application domain of a paintable computer. The foundation routines of chapter 4 were extended and combined to create four representative applications: Audio Streaming, Holistic Data Storage, Surface Bus, and Image Segmentation. Criteria that guided the selection of these four applications were that they:

- develop the algorithms and software for a practical application (either an entirely new application or a novel extension of an pre-existing one).

- demonstrate one or more elemental capabilities (communication, storage, signal processing).

- highlight important properties of the paintable architecture (price performance, compute capacity, fault tolerance).

- illustrate ways in which even contemporary applications can take on new dimensions when mapped to densely distributed hardware.

Each application was developed, tested and characterized on the Psim simulator. Target functionality was verified, both numerically and visually (using Psim s movie record option). A concluding discussion argued that, while each application could be efficiently implemented using conventional designs, the paintable-based implementation will be come commercially attractive at the point where rising operational complexity renders the alternatives impractical.

Table 5-1 summarizes the results of this chapter. For each application, there is a functional description of the application, an overview of the software implementation, a tally of noteworthy results[34] and an indexed list of the constituent pfrags.

---

34. The tallied results are printed in *italics*.

TABLE 5-1. Summary review of four applications

Application	Constituent Pfrags	Functional description / Implementation / Results
Audio Streaming	Gradient Carrier	Storage and retrieval of ordered data (using audio as example). Collective memory of particle ensemble is treated as a single monolithic memory unit, into which packetized data can be streamed. through a serial port. In storage, data is spatially and temporally decorrelated. On retrieval, data reestablishes a linear order prior to output trough a single serial port. Packetized data and associated time codes are inserted as payload into Carrier pfrags and streamed in through a single portal. In storage mode, Carriers emulate Diffusion pfrag and position themselves randomly. Retrieval is signaled by callback Gradient radiated from output portal. Carriers then compare time codes, and reorder themselves using callback Gradient for orientation. *Simplest of four applications, yet already demonstrates capacity for data storage, useful global behavior and topological independence. The example of a network-with-no-routers is characteristic of the novelty inherent in this architecture.*
Holistic Data Storage	Gradient Carrier Transform	Storage and retrieval of data augmented by "holographic" representation. Similar in context and usage to the audio streaming application, with the addition that arbitrary subsets of the stored data can be decoded back into low resolution versions of original input (behavior reminiscent of a hologram). Algorithm based on cascade of hierarchical wavelet transformation, duplication of the lowest frequency transform coefficients, and a pseudo-random 'diffusive' scattering of the transform samples throughout the memory. Implementation based on still images as input type. Images are partitioned into blocks, inserted as payload into Carriers and streamed into the particle ensemble. There, Carriers interact with pre-loaded Transform pfrags, with the effect that the time domain image data stored in the Carriers is replaced with frequency domain data. Carriers then bifurcate into multiple mini-Carriers, each containing a subset of the transform data plus a redundant copy of the lowest frequency transform component. Graphical selection of subregions of the particle ensemble via free-hand drawing. Callback Gradient extracts mini-Carriers from subregions. Frequency domain data is extracted from mini-Carriers and decoded back into image data. *Experiments confirmed holistic reconstruction, progressive refinement, and robustness to changes in the number / position of input streams. Application demonstrates paintable's capacity for both storage and signal processing.*

TABLE 5-1. Summary review of four applications

Application	Constituent Pfrags	Functional description / Implementation / Results
Surface Bus	Gradient MultiGrad Buoy Channel Coordinate	Simulation of emergent network connectivity gated by physical contact with a table. Portable compute-enabled appliances (e.g. laptops, PDAs cell-phones digital cameras) are randomly positioned on the periphery of a table (represented in simulation by a 2D particle ensemble). Devices on table exchange pfrags with the particles embedded below. These pfrags propagate and self assemble into a ring bus that serves as primary messaging conduit for the devices on the table. Optional secondary connections support concomitant processing — directed processing performed on the messages as a byproduct of transmission. Implementation based on a sample definition of peer — any external device with five portals organized in a pre-defined geometry. On contact with the particle ensemble, peers exchange pfrags with neighboring particles and sequence the formation of a multi-channel ring bus, with peers assuming the role of nodes in a multi-hop network. Optional secondary inter-peer connections with embedded local coordinate systems serve as scaffolding for distributed data flow routines. *Experimental results validated important guiding insight, namely that the growing expertise in the self assembly of complex spatial patterns can be used to direct a similarly complex flow of control and data within a distributed process.*
Image Segmentation	MultiGrad Four Experts : • Sand • Water • Sky • Tree	Region classification via competition among "experts". Simulation of a still image projected onto a 2D particle ensemble. Embedded "photosensor particles" non-uniformly sample the image and broadcast pixel data to nearby generic processing, where the pixel values appear as HomePage posts. "Expert" pfrags, each selective for a particular image feature, use mixture models to compute the conditional probability of the posted pixel value given the image feature. Experts report these probabilities as HomePage posts. Pixels are assigned to the region corresponding to the greatest conditional probability. Illustrative experiment conducted on natural image with four object classes (sand, water, sky and foliage). Four experts, each selective to one of these regions, where created using sampled training data. Segmentation performance was evaluated visually using simulator s viewer. *Incorporated parallel I/O into paintable computing environment. Additional programming examples for signal processing. Preliminary analysis of system performance placed this architecture in a price/performance regime beyond reach of contemporary centralized architectures, with the crucial caveat that the process flow in the application efficiently maps to a distributed environment.*

Wrap up

Time to step back and take stock.

The preceding chapters have distilled the essentials of a distributed architecture into a hardware reference platform, defined a programming model based on relevant abstractions, illustrated the model with simple software components, and defended the model with four application-class examples. This chapter concludes the report with a discussion of related points, a review of the contributions and suggestions for future work.

Colloquy

Over the course of this work, several related themes were regularly revisited. While none of these topics was central to the investigation, they each contributed the additional perspective that kept the larger picture in focus. This section discusses four of these topics: proximal computing, ensembles with moving particles, computational universality, and one innovator's yardstick for characterizing new models of computing.

The yardstick. In his work on neuromorphic computing systems, Mead [38] characterizes novel computing systems at three levels: *representation of information*, *computational primitives*, and *organizing principles*. When viewed through this lens, the work of this report can be described as follows:

- **representation of information**: Data is binary coded. The basic unit of information is the post on the HomePage. They are publicly visible key/value pairs capable of active/reactive behavior. The behavior is defined by an associated process fragment, which responds to adjacent posts by changing existing posts or issuing new ones.

  A useful analogy for the posts and coded behavior is the objects in an object oriented language. Here the coded pfrag behavior is analogous to object methods and the posts are analogous to publicly viewable object variables. The important caveats are that a post's visibility is local and probabilistic, and that the pfrags are spatially mobile within the computing medium.

- **computational primitives**: All information processing proceeds on Turing equivalent computing elements. The computational primitives of a *paintable* are the same as those on commonly available variants of a von Neuman machine.

  Ultimately, designers will step beyond the parochial reference platform of chapter 3 and embrace alternative computing elements based on primitives which fundamentally depart from boolean logic. As long as the computing adheres to the organizing principles listed below, many of the results from this research will carry over.

- **organizing principles**: The organizational mantra — self-assembly of small simple components into large complex aggregates — is embodied in three constructs: 1) The atomic software component is the process fragments — autonomous, mobile program entities capable of sensing and reacting to their environment. 2) pfrags signal via broadcast through a medium which supports probabilistic reception over a local area. 3) Data exchanged between proximal pfrags is used to emulate the processes of thermodynamic and scaffolded self-assembly. These processes in turn guide the positioning of the pfrags in the construction of larger processing structures.

Proximal computing. Rising performance and shrinking form factors have been hallmarks of commercial computing since its inception. And it was not long before computers were coupled to the physical environment through sensors and actuators for monitoring, analysis and closed-loop control. Shrinks in IC feature size has meant more computation in greater proximity to an expanding set of physical processes. Viewed as the next step in this progression, it is natural to ponder the

paintable's likely impact. What doors will it open? What rules will it change? What challenges will it present?

To the question of new applications, consider the perennial limiting resources for embedded computing; communication bandwidth, compute capacity, physical size, portability and reliability. *Paintable*-like ensembles, coupled with progress in MEMS, advances performance on all these fronts (save for reliability, perhaps). Three examples:

- One class of embedded control that is primarily compute-bound is predictive control systems. These are systems where the time between the arrival of the stimulus from the sensors and onset of actuation is necessarily very short, perhaps even negative (i.e. the system is anti-causal)[1]. The computing elements must maintain an updated model of the local physics. Control performance is bounded by the complexity of these models, which in turn is fixed by available compute[2].

- An application domain that is currently limited by both scale and portability is the domain of "ingestibles" — ensembles of sub $1mm^3$ modules that are ingested into a body where they coordinate to sequence through a set of functions based on chemical or mechanical actuation. Computing elements on the individual modules are necessarily modest. But a small reserve of residual capacity on each module could be aggregated to run larger procedures to direct global adaptation.

- The image segmentation app of chapter 5 is characteristic of distributed analysis techniques that are currently limited in scale, bandwidth and compute. The promise of these applications is their ability to ingest voluminous amounts of partial or ill-conditioned data, perform intensive early-stage analysis, and pass the compressed results downstream for further processing. The prototypical example from nature is the mammalian retina which builds robust, information-

---

1. This can occur in systems where the mechanical response of the actuators is very slow, or where actuation forces must be intense and prolonged.

2. Consider the case of a vehicle about to experience a destructive impact. In the instant prior to contact, a sensory enhanced central controller realizes that collapse of the frame will be unavoidable and decides how the individual members of the frame should direct their buckling. Coarse guidelines are broadcast to those components of the frame that contain fine grain embedded control, sensing and actuation. As the collision unfolds, these members adaptively fine tune their demise with the goal of minimizing injury to the vehicle occupants. In this invented extension of existing technology, the local computing elements play a fast but intense game of chess with the physics of the buckling structures.

rich representations from dense photosensor data, and then passes these representations along to the cortex with a 125× reduction in bandwidth.

In addition to more applications, there is broad consensus that increasingly proximal computing will drive fundamental changes in our programming methodologies [5] [27] [33] [38] [47]. Tennenhouse [47] sees humans assuming a supervisory role as ever faster closed-loop control systems make direct human intervention impractical. Networks of embedded controllers will communicate via packets with stochastic transit times, necessitating new design approaches for embedding these networks into the feedback loop of control systems. Lee [33] calls for the application of heterogeneous programming models — simultaneous operation of several programming models composed hierarchically within the framework of a finite state machine. In their work on "smart matter" Hogg and Huberman [27] describe organizations for multiple agents in distributed control of unstable physical systems.

The results from this report have much to contribute. The hardware reference platform extends to networked millimeter scale computing elements and supports transparent program access to heterogeneous sensors and actuators. The programming model exposes the aggregate memory and processing resources to collections of mobile program components that self-assemble into larger procedures spread over multiple particles. This suggests an intriguing scenario where the programs physically "walk" to the position of the relevant external stimulus[3].

Further work will be needed before these models can be applied directly to feedback control systems. Specifically, the stochastic nature of the interaction among the pfrags currently frustrates the analytic treatment required to insure timeliness and reliability.

Self-assembly on moving particles. There is no evidence that process self-assembly is suitable for the general case of ensembles where a subset of the particles are undergoing unconstrained motion. However there is a special case worth noting.

Recall that the positioning of the pfrags is guided by processes which are generative — which is to say that they are continually running and do not terminate when the pfrag becomes stationary. If the particles are moving at a speed which is slow compared to both the interparticle bandwidth and the internal clock speed of the

---

3. Consider a wall where someone hangs a simple antenna and the digital broadcast receiver walks to the antenna and arranges its modules over neighboring particles.

particles, then self-assembled structures will adapt and the computing should be robust. However real engineering challenges lie in the design of hardware which can deal with fluctuations in power and sporadic network connectivity as particles move along the perimeter of each other's reception area.

Computational Universality. The concept of computational universality was advanced by Alonzo Church and Alan Turing who proved that, ignoring limitations of time and memory, any computer can be programmed to emulate any other computer. The performance of any two of these *universal machines* can be related by a scalar prefactor. *Paintable* particles, with their universal computing elements interfaced to embedded local memory, are by definition computationally universal. Two basic problems with this observation are that: 1) the compute capacity of an entire particle ensemble is bounded by the capacity of a single particle, and 2) emulation is limited to programs which can fit in the memory space of a single particle.

The problem of limited memory can be alleviated through the use of three pfrag types. These pfrags interact to assemble the ensemble's collective memory into a single linear address space, that can then be addressed by the CPU of a single particle. Reads/writes to selected memory locations are performed by light weight Messenger pfrags which traverse the particle ensembles to traffic data between a CPU and a selected memory location in a distant particle. Fig. 6-1 describes the underlying mechanisms using a 1D particle ensemble for illustration.

Using this approach, *a particle ensemble can be assembled into a universal machine whose compute capacity is equal to that of a single particle and whose memory is a large fractional multiple of the ensemble's total memory.* The performance of this universal machine, relative to that of an arbitrary target machine, can be described by a prefactor. In the case of a *paintable*, these prefactors are program dependent, and group into three distinct categories. Each category is distinguished by the degree to which the executing program must recourse to Messenger pfrags to fetch instructions and data. Table 6-1 summarizes these three program groups and gives expressions for their associated prefactors.

In this table, the prefactors are defined as linear combinations of three quasi-independent prefactor components; an "architecture" prefactor (P_A), a "data fetch" prefactor (P_D) and an "instruction follow" prefactor (P_I).

- P_A The architecture component captures the architectural differences between the computing engines. When comparing the *paintable* micro to the CPU of a high performance consumer PC's, the differences reflect machine features such as clock speed, cacheing, pipeline depth, instruction set, and strategies for pre-
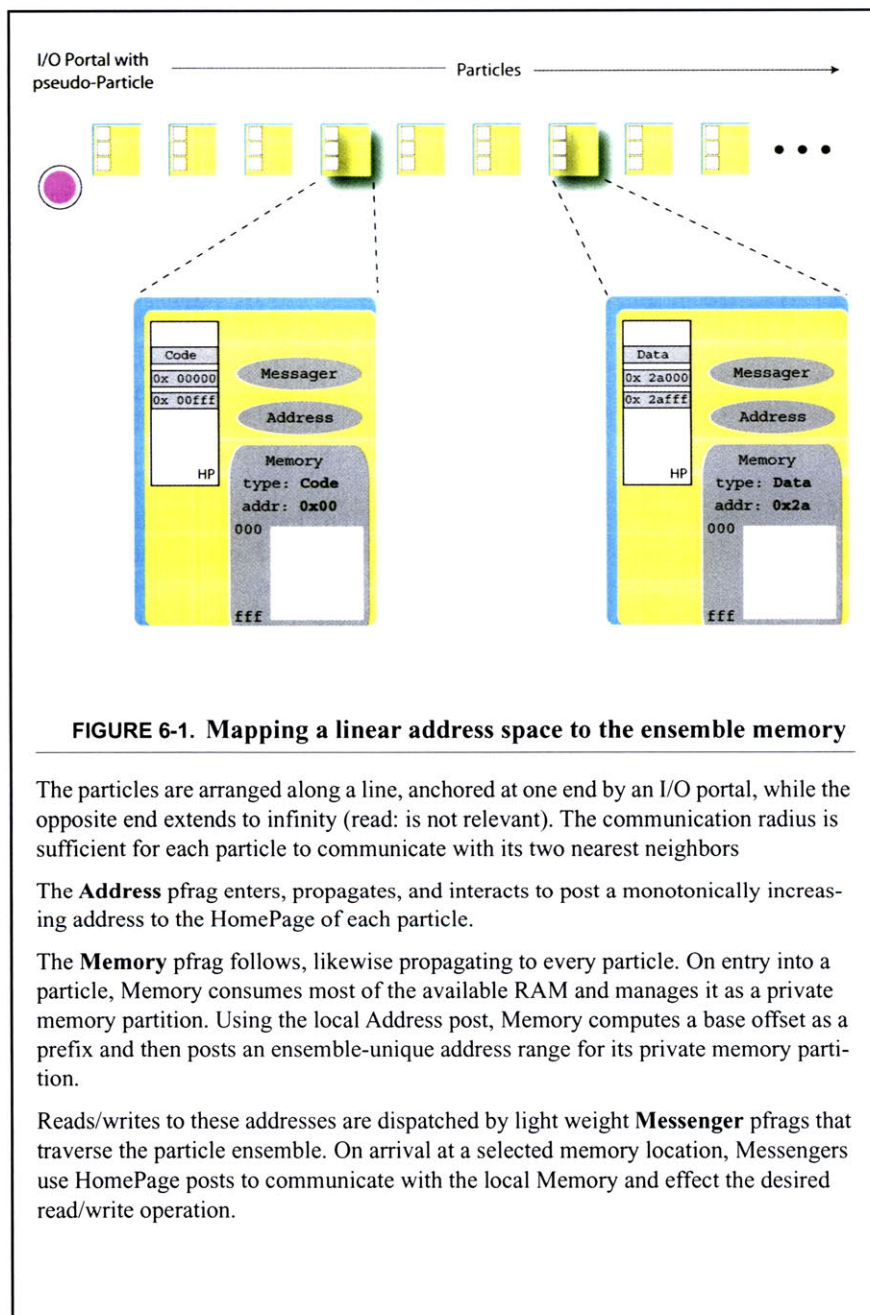
FIGURE 6-1. Mapping a linear address space to the ensemble memory

The particles are arranged along a line, anchored at one end by an I/O portal, while the opposite end extends to infinity (read: is not relevant). The communication radius is sufficient for each particle to communicate with its two nearest neighbors

The **Address** pfrag enters, propagates, and interacts to post a monotonically increasing address to the HomePage of each particle.

The **Memory** pfrag follows, likewise propagating to every particle. On entry into a particle, Memory consumes most of the available RAM and manages it as a private memory partition. Using the local Address post, Memory computes a base offset as a prefix and then posts an ensemble-unique address range for its private memory partition.

Reads/writes to these addresses are dispatched by light weight **Messenger** pfrags that traverse the particle ensemble. On arrival at a selected memory location, Messengers use HomePage posts to communicate with the local Memory and effect the desired read/write operation.

TABLE 6-1. Program types and associated prefactors

for programs where ...	the expression for the prefactor is ...
• all instructions and data fit into a single particle. • no need for Messenger pfrags	P_A
• all the instructions fit into a single pfrag. • data is distributed through memory of neighboring particles • Messenger pfrags fetch data	$P_A \times P_D$
• both instructions and data stored in multiple particles • Messenger pfrags fetch data • execution "focus" moves among particles following instruction stream	$P_A \times (P_D + P_I)$

dictive execution. Assuming that the paintable micro was a 16 bit machine spec'ed down for small size, and power consumption, the expected value for the prefactors will lie in the range of 75-200.

For instances when all the code — instructions, data, and scratch space — fit into a single particle, the prefactor is completely defined by the P_A component.

- P_D The "data fetch" component expresses the penalty incurred for access to memory in other particles. This component is relevant for programs where the instruction stream fits into a single particle, but the data must be stored in the memory of multiple particles — typical of a short program running on a large data set. In this case, data must be fetched by migrating Messenger pfrags that travel to the selected particle and negotiate access to the memory. Simple writes require a one way trip. Reads and acknowledged writes require a round-trip.

P_D is computed as the product of three terms: the expected frequency of access to off-particle data, the average number of hops for the Messenger, and P/N — the ratio of the processor clock speed to the network bandwidth. Qualitative estimates for the terms would be 1/2 for the fraction of instructions involving memory access, N_p (the number of particles in the ensemble) for the expected number of Messenger hops[4], and a P/N ratio on the order of 10^3.

---

4. Assumes round-trip Messenger travel, and access to data that is uniformly distributed over the memory space of a 1D ensemble of Np particles: $2 \times (N_p / 2)$.

- **P_I** The "instruction follow" component estimates the penalty for transferring the execution of a program from the CPU of one particle to the CPU of another. For programs where the instruction storage spans multiple particles, an alternative to a steady progression of instruction fetches is to move program execution to the particle containing the current instruction. Transfer of the execution "focus" would be facilitated by a pfrag that encapsulated the processor state.

The frequency of this transfer-of-focus will depend strongly on the program structure. However, baring any unfortunate pathologies[5], any program whose instruction stream exhibits reasonable spatial coherence will have a value for P_I that is negligible compared to the value for P_D.

Using these expressions for the prefactor components, we can coarsely compare the CPU of a single particle to a full featured CPU typical of a high end consumer PC. The target program has an instruction stream that fits into the memory of three particles and operates on 4 MB of data that is evenly distributed over 97 particles. Using the component estimates to the right, the resulting prefactor is on the order of 10^7 !!

$$P_A = 200$$

$$P_D = \frac{1}{2} \times 100 \times 10^3$$

$$P_I \ll P_D$$

Unless you are a pure theorist, this is depressing news. Two architectures consuming roughly comparable amounts of silicon still exhibit a multiplicative performance difference on the order of ten million. Two equally valid responses to this bad news are:

1. Fearless engineering. Close the gap by dividing the problem up into its root causes and addressing them individually.

2. Pragmatic morphology. Admit that dolphins do not fly and that distributed hardware is only really suitable for distributed computing.

Our fearless engineer has grounds for optimism. The single biggest contributor to the prefactor is the disproportionately slow network bandwidth. Balancing the inter-particle transfer bandwidth with the CPU clock speed [6] would improve the

---

5. Like an inner loop crossing a particle memory boundary

6. One possible approach would be to base the inter-particle communication on optical carriers — fitting the particles with components to emit and sense light and relying on the translucency of the medium to limit propagation.

prefactor by an order of 10^3. Additional potential for improvement lies in the dynamic reordering of the data stored in the memory of the peripheral particles — those particles that are serving the memory accesses for the particle that is hosting the processing. Borrowing from conventional caching schemes, peripheral particles could be constantly exchanging their Memory pfrags looking to minimize the average number of particle hops for access to data. Finally, while it is actionable heresy, one could imagine confining the processing to CPU's with enhanced capacity — either full featured CPU's embedded in I/O portals that make network contact to the particle ensemble, or a collection of "super particles" that have been interspersed among the generic particles. Here, in exchange for his soul, the designer could expect up to 100× additional improvement in the prefactor, depending on when the networked data access became a bottleneck.

The morphologists face a more uncertain path. On the one hand, they know instinctively that distributed hardware performs at near-full capacity on distributed computation, and that the performance gains can be dramatic. On the other hand, there is still no such thing as a "universal distributed computer". Distributed computing still lacks a comprehensive descriptive framework with the formal power of Turing's work. And to date, there is neither a generic architecture for universal distributed computing nor a statement of equivalence across architectures. Absent this framework, investigators recourse to the somewhat ad hoc algorithm selection of this research, and the more insightful approaches exemplified by Spears and Gordon [45] who studied the commonalities between distributed control and distributed computation in the context of "artificial physics".

Contributions

The contributions come in two flavors; those that are tangible, immediate and provable, and those that are suggestive and need time to germinate. The tangible contributions in summary are:

- *Process self-assembly*: a novel distributed programming methodology which maps existing techniques in material and virtual self-assembly to a broad class of dense ensembles of asynchronous, locally inter-networked computing nodes.

- A programming model built around a novel abstraction for inter-process communication, and the construct of a "process fragment" as the atomic element of process self assembly.

- The concept of a "vfrag" and "operators" as basic instances of abstraction and modularity in process self-assembly.

- Four applications that are both novel in their own right and that are representative of a broader class of useful algorithms.

The larger, hoped-for contribution has been an instance of consciousness raising within the engineering community. Although the situation is improving, self organization is not regarded yet a fully vested member of the engineers' tool kit. This work has argued that some degree of self organization is necessary if one would engage the huge compute capacity of a *paintable* at its unique price point. To the degree that the applications have demonstrated compelling behavior with acceptable performance bounds, this dissertation should lend credence to the use of self organization as an engineering tool.

Future Work

The field is young, ripe with promise and full of fun waiting to be had. Three fruitful avenues for continued research are hardware, computational theory, and advanced applications.

Hardware. Simulation will only get us so far. And in development of parallel computing systems, the immense speed differential between real hardware and virtual emulation can be extremely limiting. Fortunately, this does not mean that we have to wait for the paint to chip. Interim solutions involving pushpin-style hardware with approachable technology thresholds will provide an effective bootstrap for growing a community of early investigators. Near term development will focus on 2D ensembles, with investigators ultimately progressing to 3D structures.

A related area, as important as the particles themselves, is the design of the I/O portals, particularly ones with a size and form factor comparable to that of a particle. Particle-sized sensors and actuators will be key to exploiting the parallelism of a *paintable*, and for finely controlling the ratio of I/O to processing.

Finally, there will be a frequent need for a display in which each particle can individually address a small number of pixels. Work on the pushpins suggests an elegant construct based on e-Ink [13] [30]. The planar composite is fitted with three additional layers: a second resistive sheet, a layer of e-Ink, and an outer layer of clear conductive material such as ITO (fig. 6-2). Pins issuing from particles assert a
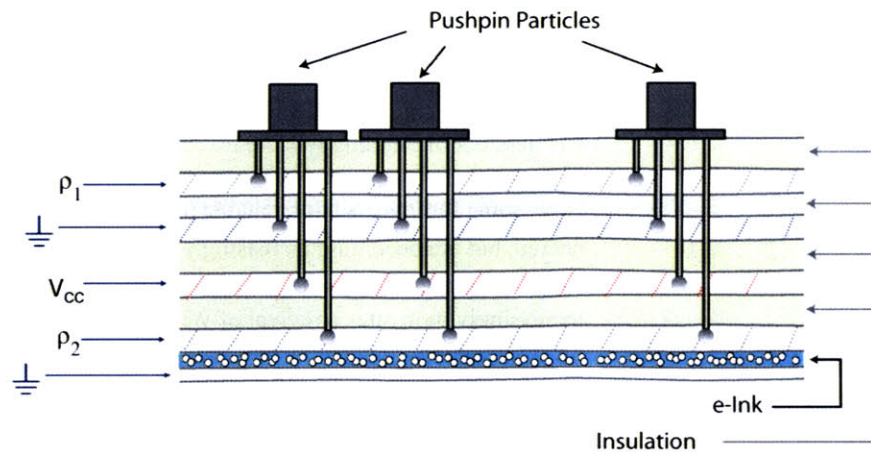
Pushpin Particles

ρ_1

V_{cc}

ρ_2

e-Ink

Insulation

FIGURE 6-2. Pushpins extended for e-Ink Display

The pushpin concept of figure 3-2 is extended to incorporate an additional pin for display. The display pin (longest) creates a local potential in a second resistive medium that supports a sharp ohmic drop off. This potential produces a field across e-Ink spheres that are biased to ground by a outer layer of clear conductive material (e.g. ITO).

potential on the second resistive sheet, where the material properties cause the potential to roll off sharply. Areas of nonzero potential result in a field across the e-Ink plane, terminating in the outer layer which has been biased to ground. By regulating the potential, particles can exercise gray level control over a small portion of the display.

Computation Theory. Is there any way to formally relate the compute capacity of an error-free Turing equivalent machine to the aggregate capacity of networked nodes, where both the network and the nodes are subject to a nonzero probability of failure?

Through the mid 60's, there were credible arguments that commercial manufacture of reliable computing elements exceeded basic engineering limits. During this period, Winograd and Cowan[51] began to develop the mathematical formalisms for describing the computation capacity of a collection of faulty nodes that communicated over a faulty network. In results with intriguing parallels to the information theoretic definition of channel capacity, they showed that increasing redundancy in

such a network produced threshold behavior in the reliability of program execution. In other words, the redundancy of the hardware could be raised to a point where program execution could be provably reliable even on an ensemble containing faulty nodes and interconnects. This promising research laid important groundwork, but was prematurely undermined when its driving assumption proved untrue — namely when Univac, IBM, Control Data, and their ilk actually did make a going business out of building reliable machines.

Now, some forty years later, reliable high end computing elements can still be engineered, but are becoming increasingly difficult to pay for[7]. The widening price / performance advantage of distributed computing on alternative substrates[8] increasingly motivates a revival of Winograd and Cowan's approach. My hope is the research reported herein adds to this momentum.

Advanced Applications. The grand challenge of the *paintable* is the search for applications that make sensible use of its copious compute capacity. These advanced applications will come in three flavors:

· things that we can do now, but would like to do faster.

· things that we've always wanted to do, but that were never practical on a serial architecture.

· and things that (almost) no one has ever thought of.

The first group consists of distributed extensions of common serial algorithms. Examples include mainstay algorithms for control, classification and signal processing. Here, extensions from serial machines to the *paintable* are guided by the oft-cited tenet of parallel computer design — for a parallel machine to be useful, the topology of the machine must be well matched to the natural topology of the problem. Given a mature algorithm with a well understood data flow topology, the common approach will be to employ self organization on the *paintable* to mirror this preferred topology in the data flow among the particles[9].

---

7. The cost referred to here is the cost of fab for producing IC's with the smallest feature sizes. US$ 2B was a frequently cited estimate for the cost of constructing a fab from scratch for 0.18 micron chip manufacture.

8. Computing on biological, chemical, molecular and atomic substrates.

9. This basic strategy also guides compiler design for commercial parallel systems — usually those employing coarse grain parallelism such as VLIW machines.

The second group consists of applications whose execution was first made practical by the advent of densely distributed hardware. Typical applications are those where the characteristics of the environment mandate the autonomy of the computing elements. Examples from this group are applications that involve dynamic estimation of node position, where the nodes use the position info to specialize their function and coordinate their activity. Two examples:

·   **recovery of 3D surface geometry** An object of arbitrary shape has nodes semi-uniformly distributed over its surface. The nodes interact locally to estimate the 3D shape. Algorithms for this have been proposed [23] but are still immature.

·   **sky art** Nodes, carried as payload in a fireworks shell, are exploded outward from the shell and distributed over a large volume of space. On the way down, the nodes estimate their relative 3D coordinates, use their coordinate to select a pixel intensity (based on a pre-stored image), and burn with the assigned color[10].

The third type of application are those for which we still have few principled insights, but a vague awareness of a looming problem. These applications are by definition highly speculative, but often they are the ones that generate the fresh perspectives that are crucial to progress. As a parting indulgence, I would like to elaborate on an example that I regard as important; collaborative problem solving on a smart surface.

In traditional collaborative problem solving, two or more people pool their skills, knowledge, and material resources to collectively arrive at a solution to the problem. While frustrated by corporal limitations, would-be collaborators seek to transmit to their partners a snapshot of everything they know that might be of relevance to the problem. Many cultures capture this ideal of transparency in phrases like "putting your heads together".

In the smart surface variant, we would like the minds to meet in the table. The environment is marked by meeting tables with teraFLOP embedded compute capacity[11] and wearable memory storage in excess of a terabyte. This memory is owner-centric, with portions of the data purposefully keyed in while other portions are gathered automatically by on-body sensing devices — in many ways, an extension of the

---

10. Few cities are as earnest about their July 4th observation as the city of Boston. Ideas like kilometer wide flags as sky-art seeded from firework shells come to you just by breathing the air here. I can't wait to move to the suburbs.

11. Table dimensions: 5ft x 2ft   Particle density: 14 particles / in^2 Clock speed: 50MHz.
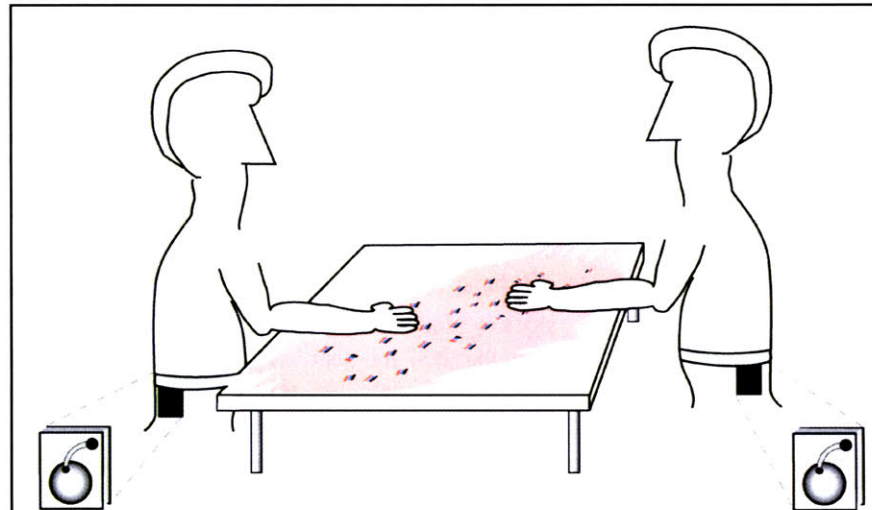
FIGURE 6-3. Collaborative Problem Solving on a Smart Surface

Two people, each fitted with a terabyte of wearable memory, approach a table fitted with a teraFLOP compute in order to jointly search for a solution to a problem. Information on their disks is represented as a collection of autonomous packets (i-packets). A high bandwidth channel from each person's disk into embedded particle ensemble is gated via natural gestures such as the physical contact with the hand. Once in the table, the i-packets interact to progress toward a solution, with the human observers playing a supervisory role to guide the interaction.

owner's biological memory. In collaboration, the goal is to use the processors embedded in the table to constructively combine the information from the disks. The problems are that there are too many particles to direct each individually, and that the size of the memory frustrates any taxonomic familiarity with its contents. In other words, we can't select the data, and we can't micro-manage the processing. A solution would be to somehow communicate a well posed problem to the table, randomly diffuse the contents of the disks into the table, let the elements of the data interact, and restrict the humans to a supervisory role for guiding that interaction in the search for a solution.

Science fiction? Hardly. In a restricted sense, engineers and scientists do this every day. Commercial software packages for symbolic mathematics (e.g.. Mathematica) use rule-based search techniques similar in spirit to the scheme outlined above. A

user types in a symbolic expression of near arbitrary complexity. The implied "problem" is to reduce the expression to its simplest form. In its *eval* loop, Mathematica iteratively applies a set of rules to systematically reduce the complexity of the expression, halting either when the expression is in a canonical form or when further attempts at reduction fail. The crucial point is that all the rules are checked at each round of iteration, and that the rules are independent. While today the rule set is bundled with the software, it can be (and initially was) collected from a set of disparate sources.

The extension to the general case of collaborative problem solving on a paintable begs two questions:

1. Is the pfrag a useful representation of knowledge?
2. Can the organizing principles that guide the assembly of the pfrags also be used to create structures and representations that aid in problem solving?

These questions require much more additional research. Yet we need not wait until we can pattern a brain before we can make sensible use of this computing. Distributed problem solving by ecologies of cooperating agents has been extensively studied in artificial intelligence [18] [16], with strong progress made on those problems that can be expressed as searches through large problem spaces. In the special case of heuristic search operating on constraint satisfaction problems, Huberman, et.al. [12] [25] presented an analytic treatment of cooperative search and quantitatively established the performance gain as a function of the ecology's size and diversity. In their application of machine understanding to news editing, Gruhl and Bender [24] demonstrated how a collection of disparate agents acting independently can already be used to catalog a large corpus of news feeds and perform efficient query-by-example searches. Studies such as these are suggestive of the potential of a *paintable*-based application, but also underscore the difficulties lurking in the implementation.

So why the fascination with collaborative problem solving? Why not space exploration, geophysics, cryptography or molecular biology? Because human-to-human communication underlies virtually all organized social behavior — with disastrous consequences when it fails. And if the processes of human communication ever do find expression in the formalisms of computer science, then I believe that collaborative problem solving will emerge as one of the field s NP-complete problems.

And we all like a challenge.

Wrap up

Bibliography

[1] Alien Technology Corporation *Fluidic Self Assembly: White Paper*, http://www.alientechnology.com/d/technology/pdf_library.html, 1999.

[2] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss *Amorphous Computing* Communications of the ACM, vol. 43, pp. 74-82, 2000.

[3] A. I. Akinwande, D. G. Pflug, and B. Johnson, *Cone and lateral thin-film field emitter displays*, presented at IEEE International Conference on Plasma Science, 1998.

[4] D. H. Ballard, *An Introduction to Natural Computation*, Cambridge, Massachusetts: The MIT Press, 1997.

[5] A. A. Berlin, *Towards Intelligent Structures: Active Control of Buckling*, Ph.D. in Electrical Engineering and Computer Science. Cambridge, MA: Massachusetts Institute of Technology, 1994, pp. 1-101.

[6]   M. Blumrich *Virtual Memory Mapped Network Interface for the Shrimp Multicomputer* presented at 21st Annual International Symposium on Computer Architecture, 1994

[7]   N. B. Bowden, *Order and Disorder: Mesoscale SelfAssembly and Waves*, Ph.D. in Department of Chemistry. Cambridge, Massachusetts: Harvard University, 1999, pp. 307.

[8]   A. M. Bruckstein, R. J. Holt, and A. Netravali, *Holographic Representation of Images* IEEE Transactions on Image Processing, vol. 7, pp. 1583-1597, 1998.

[9]   W. Butera and V. M. Bove. Jr., *The Coding Ecology: Image Coding via competition Among Experts*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 10, pp. 1049-1058, 2000.

[10]   R. Castagno, T. Ebrahimi, and M. Kunt, *Video Segmentation Based on Multiple Features for Interactive Multimedia Applications*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 8, pp. 562-571, 1998.

[11]   E. Chalom, *Statistical Image Sequence Segmentation Using Multidimensional Attributes* Ph.D in Department of Electrical Engineering and Computer Science. Cambridge, Massachusetts: Massachusetts Institute of Technology, 1998, pp. 201.

[12]   S. H. Clearwater, B. A. Huberman, and T. Hogg, *Cooerative Solution of Constraint Satisfaction Problems*, Science, vol. 254, pp. 1181-1183, 1991.

[13]   B. Comiskey, J. D. Albert, H. Yoshizawa, and J. Jacobson, *An electrophoretic ink for all-printed reflective displays*, Nature, vol. 394, pp. 253-255, 1998.

[14] D. Coore, *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer* Ph.D in Department of Electrical Engineering and Computer Science. Cambridge, Massachusetts: Massachusetts Institute of Technology, 1999, pp. 295.

[15] D. D. Corkill, "Blackboard Systems," AI Expert, vol. 6, pp. 40-47, 1991

[16] E. H. Durfee and T. A. Montgomery, *Coordination as Distributed Search in a Hierarchical Behavior Space*, IEEE Transactions on Systems, Man, and Cybernetics, vol. 21, pp. 1363-1374, 1991.

[17] R. Engelmore and T. Morgan, *Blackboard Systems* Adison Wesley, 1988

[18] L. Gasser and M. N. Huhms, *Distributed Artificial Intelligence*, vol. 2. Menlo Park, CA.: Morgan Kaufmann, 1989.

[19] D. Gelernter, *Generative Communication in Linda* ACM Transactions on Programming Languages and Systems, vol. 7, pp. 80-112, 1985

[20] N. Gershenfeld, *The Nature of Mathematical Modeling* Cambridge University Press, 1999 (see Chapter 7, section 2 on Parabolic Equations: Diffusion)

[21] M. Gorbet, M. Orth, and H. Ishii, *Triangles: Tangible Interface for Manipulation and Exploration of Digital Information Topography*, presented at Proceedings of Conference on Human Factors in Computing Systems (CHI '98), Los Angeles, 1998.

[22] D. H. Gracias, J. Tien, T. L. Breen, C. Hsu, and G. M. Whitesides, *Forming Electrical Networks in Three Dimensions by Self-Assembly*, Sciene, vol. 289, pp. 1170-1172, 2000.

[23] T. Graepel and K. Obermayer, *A Stochastic Self-Organizing Map for Proximity Data*, Neural Computation, vol. 11, pp. 139-155, 1999.

[24] D. Gruhl and W. Bender, *A new structure for news editing*, IBM Systems Journal, vol. 39, pp. 569-588, 2000.

[25] B. A. Huberman, *The Performance of Cooperative Processes*, Physica D, vol. 42, pp. 38-47, 1990.

[26] C. Hewitt, *Description and theoretical analysis (using schemata) of PLANNER, a language for proving theorems and manipulating models in a robot*. Ph.D. in Department of Mathematics. Cambridge, Massachusetts: Massachusetts Institute of Technology, 1971, pp. 346.

[27] T. Hogg and B. A. Huberman, *Controlling Smart Matter*, Smart Materials and Structures, vol. 7, pp. R1-14, 1998.

[28] C. Intanagonwiwat, R. Govindan, and D. Estrin, *Directed Diffusion: A Scalable Robust Communication Paradigm for Sensor Networks*, IEEE Personal Communications, Special Issue on "Smart Spaces and Environments", vol. 7, pp. 28-34, 2000.

[29] R. J. Jackman, S. T. Brittain, A. Adams, M. G. Prentiss, and G. M. Whitesides, *Design and Fabrication of topologically Complex, Three-Dimensional Microstructures*, Science, vol. 280, pp. 2089-2091, 1998.

[30] J. Jacobsen, B. Comiskey, B. Turner, J. Albert, and P. Tsao, *The last book*, IBM Sytems Journal, vol. 36, pp. 457-463, 1997.

[31] M. Jovanovic and V. Milutinovic, *An Overview of Reflective Memory Systems* IEEE Concurrency, vol. 7, pp. 56-64, 1999.

[32] Y. G. Leclerc, *Constructing Simple Stable Descriptions for Image Partitioning*, International Journal of Computer Vision, vol. 3, pp. 73-102, 1989.

[33] E. A. Lee, *Embedded Software*, University of California at Berkeley, Berkeley, CA UCB ERL Memorandom MO1/26, July 12, 2001 2001.

[34] J. Lifton, *Pushpin Computing: A Platform for Distributed Embedded Ubiquitous Computing* Masters in Media Arts and Sciences (in preparation). Cambridge, Massachusetts: Massachusets Institute of Technology, 2002.

[35] D. Marr, *Vision* W. H. Freeman and Company, 1982.

[36] S. Matsuoka and S. Kawai, *Using Tuple Space Communication in Distributed Object-Oriented Languages* presented at OOPSLA '88, 1988

[37] J. D. McLurkin, *Algorithms for Distributed Sensor Networks*, MS in Department of Electrical Engineering and Computer Science. Berkeley, California: University of California at Berkeley, 1999, pp. 1-50.

[38] C. Mead, *Neuromorphic Electronic Systems*, Proceedings of the IEEE, vol. 78, pp. 1629-1636, 1990.

[39] M. Minsky, The Society of Mind: Simon & Schuster, 1986.

[40] R. Nagpal, *Programmable Self-Assembly: Constructing Global Shape using Biologically-inspired Local Interactions and Origami Mathematics* Ph.D in Department of Electrical Engineering and Computer Science. Cambridge, Massachusetts: Massachusetts Institute of Technology, 2001, pp. 105.

[41] R. Nagpal, *Organizing a Global Coordinate System from Local Information on an Amorphous Computer*, Massachusetts Institute of Technology, Cambridge, Massachusetts, A.I. MEMO 1666, August 12, 1999 1999.

[42] D. A. Norman, *The Design of Everyday Things*. New York: Basic Books Inc., 1988.

[43] M. Resnick, *Turtles, Termites and Traffic Jams: Explorations in Massively Parallel Microworlds* MIT Press, 1994.

[44] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, *Ant-based Load Balancing in Telecommunication Networks,*.

[45] W. M. Spears and D. F. Gordon, *Using Artificial Physics to Control Agents*, presented at International Conference on Information Intelligence and Systems, Los Alamitos, CA, USA, 1999.

[46] G. Strang and T. Nguyen, *Wavelets and Filter Banks*, Wellesley, MA: Wellesley-Cambridge Press, 1996.

[47] D. Tennenhouse, *Proactive Computing*, Communications of the ACM, vol. 43, pp. 43-50, 2000.

[48] B. Ullmer and H. Ishii, *Emerging Frameworks for Tangible User Interfaces*, IBM Systems Journal, vol. 39, pp. 915-931.

[49] J. Underkoffler and H. Ishii, U*rp: A Luminous-Tangible Workbench for Urban Planning and Design*, presented at Proceedings of Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, Pennsylvania USA, 1999.

[50] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister, *Smart Dust: Communicatng with a Cubic-Millimeter Computer* Computer, vol. 34, pp. 44-51, 2001.

[51] S. Winograd and J. D. Cowan, *Reliable Computation in the Presence of Noise*. Cambridge Massachussets: MIT Press, 1963.

[52] L. Wittie and C. Maples, *Merlin: Massively Parallel Heterogeneous Computing* presented at International Conference on Parallel Processing (ICPP-89), 1989

[53] L. Wittie, G. Hermannsson, and A. Li, *Eager Sharing for Efficient Massive Parallelism* presented at International Conference on Parallel Processing (ICPP-92), 1992.

[54] S. C. Zhu and A. Yuille, *Region Competition: Unifying Snakes, Region Growing, and Bayes/MDL for Multiband Image Segmentation*, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 18, pp. 884-900, 1996.

Yield, Cost and Die Size

This appendix employs a simple numerical study to illustrate the dependencies between manufacturing costs, die size, and density of process failures. Two assumptions govern; 1) the manufacturing costs are linearly related to the total area of the wafer from which operable dies are harvested, and 2) process failures have limited spatial extent that are typically small compared to the area of a single die. When a portion of a die is affected by a process failure, the entire die is regarded as inoperable. Use of smaller dies naturally limit the financial loss incurred by each failure. This appendix quantitatively examines this dynamic using two models of process failure operating on three characteristic die sizes.

Each test collects comparative data for three characteristic dies sizes; 1 cm^2, 25 mm^2 and 1 mm^2 (large, medium, and small). The dies are arranged in a rectangular lattice over the active area of an 8-inch wafer[1] (fig. A-1). The 14cm x 14cm rectangle contains space for 196 large dies, 784 medium dies and 19,600 small dies.

----

1. Substantial liberty is taken with the term "active area". In practice, the active area of a processed wafer more closely approximates a circle. But the dies are still arranged on a rectangular lattice. Approximating the active area as a rectangle supports a reasonable examination of the failure dynamics while simplifying the calculations.
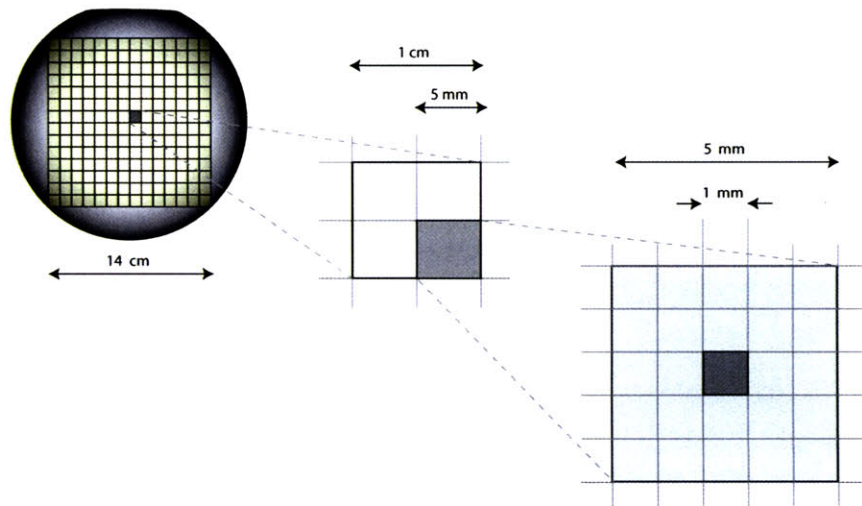
FIGURE A-1. Relative die sizes

Three die sizes are considered in these tests: a 1cm^2 die, a 25 mm^2 die and a 1 mm^2 die. In each test, the dies are assumed to be positioned in a rectangular lattice covering the 14 cm x 14 cm active area of a wafer.

In each test run, failures are randomly positioned on the wafer and the affected dies are tagged as inoperable. As failures accumulate, the total area of the operable dies is tabulated for each size scale. Tests are repeated 1000 times to account for the variance due the random placement of the failures.

Point Failures. The first test considers the case of process failures whose footprints are small compared to the area of even the smallest dies. Defects are modeled as point failures which disable one and only one die at each scale. As the test proceeds, the failures are randomly positioned on a rectangular grid with a spacing of 0.5 mm^2 and an initial offset of 0.25 mm (illustration). For each new failure, the position of the failure is used to determine which of the dies at each size scale has been rendered inoperable[3], those dies are removed from the pool of functioning dies and the fraction of operable wafer is retabulated for each die size.
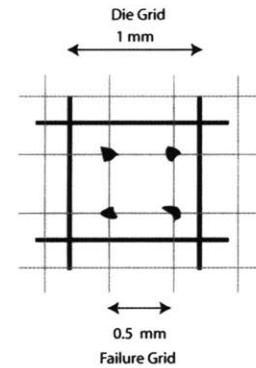

Die Grid
1 mm

0.5 mm
Failure Grid

Fig. A-2 plots the yield as a function of failure density for each of the three die sizes. Yield values represent the fraction of the wafer occupied by functional dies. The plot format is log-log. Predictably, the yield is independent of the die size at the extremes of failure density. However in the interim region where defects accumulate, the relation between yield and defect rate depends strongly on the die size.

Table A-1 lists values for selected defect rates. Note that as the yield of the large dies passes through the 30 %, the yield of the small dies is still almost unity. Likewise, when the large die yield has dropped to the point where the wafer produces an average of only one functioning die, the difference in yields across die size approaches 200 fold.

TABLE A-1. Yields for selected defect rates

No. of Defects (label from plot of fig. A-2)		Yield areal fraction (number of functioning dies)					
		Small dies		Medium dies		Large dies	
220	(x_1)	0.989	(19,380)	0.755	(592)	0.326	(64)
600	(x_2)	0.970	(19,006)	0.464	(364)	0.048	(9)
1060	(x_3)	0.947	(18,560)	0.256	(201)	0.005	(1)

---

2. Each 1mm^2 area therefore contains four failure points.

3. large dies could already be inoperable due to a previous failure

Areal Failure. In this second test, process failures have a spatial extent, with each defect effecting an area of up to 1/4 the area of a small die. Defects are positioned on a 2D grid with a spacing of 0.5 mm and an initial offset of 0.25 mm (illustration). The errors are therefore represented as a lattice with 4x the density of the small die lattice and a relative offset. At the 1mm die scale, a quarter of the failures effect only one die, one half of the failures cross the boundaries of two dies and the remaining quarter affects four adjacent dies. This corresponds to an expected loss per failure of 2.5 dies at the small scale, 1.21 at the medium scale and 1.1 at the large scale.

Die Grid
1 mm

0.5 mm
Failure Grid
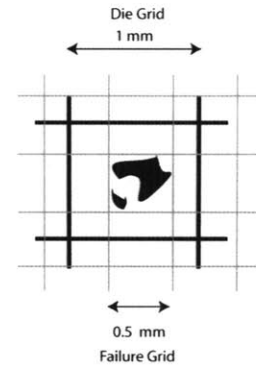
As in the previous test, defects are positioned randomly on the grid. For each new failure, the position of the defect is used to select those dies that have been rendered inoperable. The area of remaining dies is used to tabulate the instantaneous yield. The procedure is repeated for each of the three die sizes. Fig. A-3 shows the data plotted against log-log axes. As before, the relationship between yield and defect density is a governed by the die size. Here the robustness of the small dies is somewhat reduced by the fact that single errors preferentially effect multiple dies at the smallest scale. However the worst case difference in yield still approaches 200 fold[4].

Discussion. The insights from these two tests are moderated by the fact that real process failures occur simultaneously in multiple modes, some of which are difficult to model. Nevertheless, the disparity in device yields as a function of die size is a first order effect that transcends the simplifications adopted here. This phenomena manifests itself every day in the relative price of small embedded controllers and high end CPU's.

---

4. the 200x figure conveniently ignores the case where the defect rate is so high that wafers produce no working large dies.
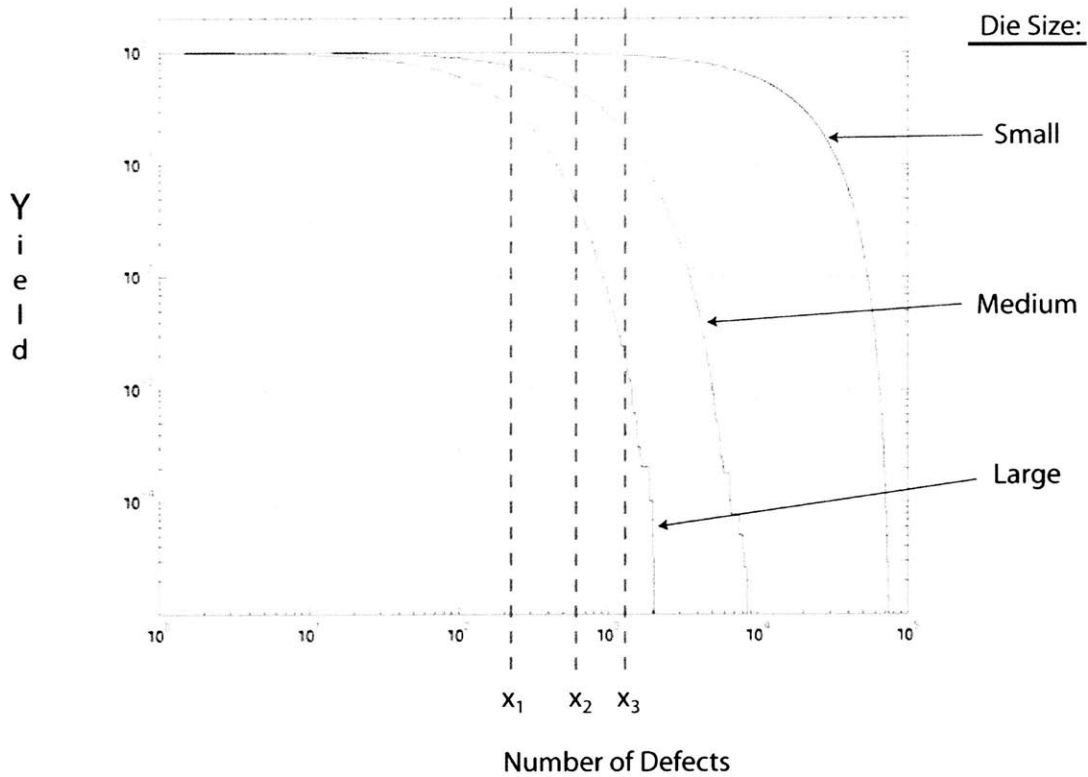
FIGURE A-2. Yields as a function of the density of point failures

Process yield is plotted as a function of the defect density for three die sizes: 1 cm^2, 25 mm^2, and 1mm^2. The yield corresponds to the fraction of the wafer that is occupied by functioning dies. The results were generated by averaging data from 1000 test runs (hence the yields corresponding to fractional dies sizes). The plot format is log-log.

In the intermediary region of fractional defect density, sensitivity of the yield to the defect rate is strongly affected by the die size. An extreme of this occurs at the defect rate labelled x_3 Here, the large die yield of 0.5 % corresponds to one working die. Yet the small die's yield is 94.7 % — almost a 200 improvement.
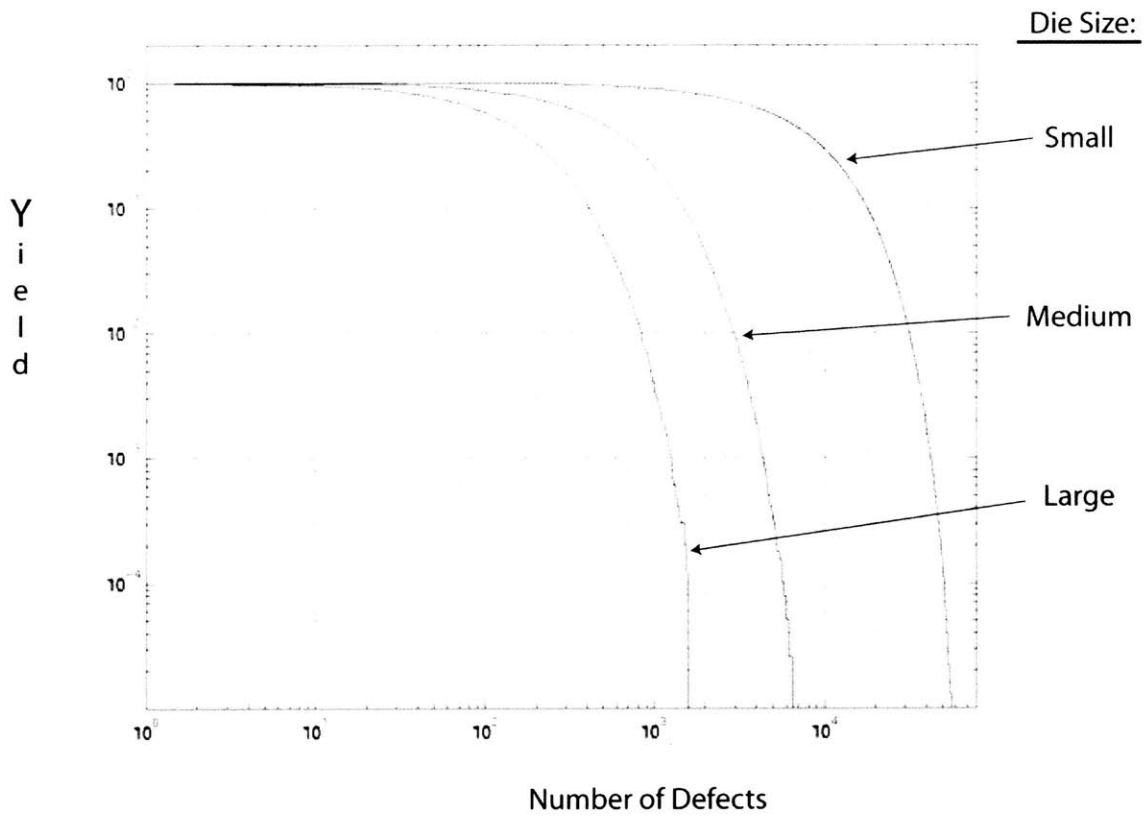
FIGURE A-3. Yields as a function of simple areal failures

In this test, process failures have a large spatial extent and are capable of disabling multiple dies. The tests otherwise parallel the procedures followed in the tests on point failures, with comparable results.

5216-19