Programming a Paintable Computer

a PhD Thesis Proposal

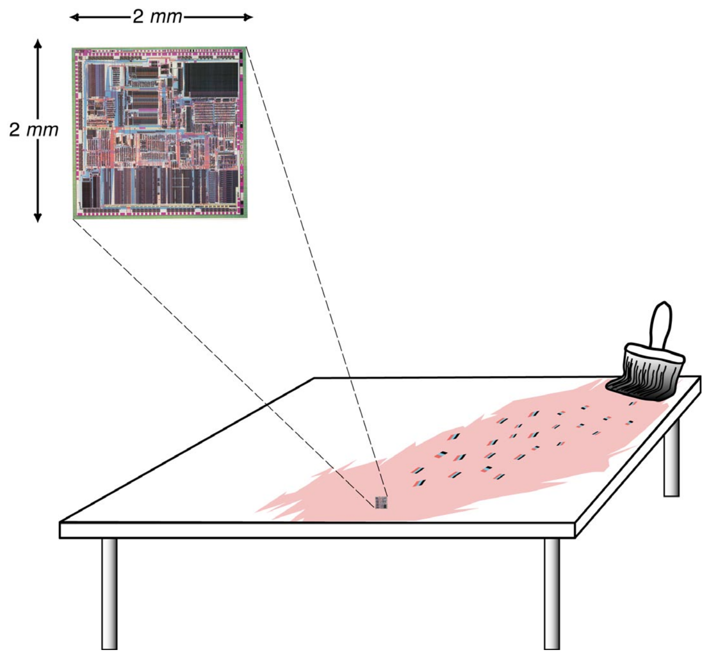
William Butera

ABSTRACT

The long term goal is to create a 'paintable computer' — an instance of several thousand copies of a single integrated circuit (IC), each the size of a large sand kernel, uniformly distributed in a semi-viscous medium and applied to a surface like paint. Each IC contains an embedded micro, memory and a wireless transceiver in a 4 mm^2 package, is internally clocked, and communicates locally.

While the hardware presents its own challenges, the deeper problems lie in the programming model. In this research, we develop a candidate programming model and qualify its performance over a set of representative applications. Work begins with a hardware reference model for the individual computing particles. A first cut programming model is proposed and initial applications are developed. Results from the applications are fed back to drive an iterative refinement of the programming model, followed by a succeeding rounds of application development.

Preliminary thesis statement: "A programming model employing a self-organizing ecology of mobile code fragments supports a variety of useful applications on a paintable computer"



* Can be skipped in an abbreviated review of the document

1 Introduction

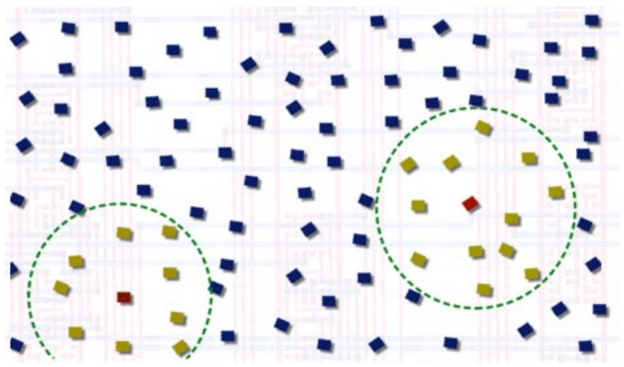
Scenario — Painting the computing

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 furniture, clothing and random surfaces.

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, this thesis pursues the notion of a *paintable computer* — an instance of a pinless IC with an on board micro, 50K of memory and a wireless transceiver, all shrunk down to the size of a small match head and powered parasitically. Several thousand of these



In a *paintable* computer, miniaturized low power micro's are pseudo randomly distributed over a surface.

Communication, supported by wireless links, is limited to the local neighborhood.

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 are identical to the transceivers on 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 largest question marks are hanging over the programming model.

For many, this level of miniaturization and transparency seems fanciful. But for manufacturers in the allied fields (mixed signal CMOS, MEMs, sensors, optics), the path toward the requisite manufacturing capabilities seems natural, indeed in many cases almost pre-ordained. And yet, the uncertainty persists. Indeed, it is exactly those manufacturers — who for years have seen pieces of the production puzzle falling into place — who have always recoursed to the question “How would anyone ever program such an ensemble?”.

Back in the lab, visions of ultra-dense computing have been doing the rounds for some years now under monikers like moletronics[5], smart dust[3], and amorphous computing[1]. Topics range from complete architectures for distributed sensing, to simple computing on biological, molecular and atomic substrates.

In my view, the most important component of the work is not the development of the enabling hardware. Rather, it is the definition of how non-specialists will use this form of computing. When a 4’x4’ slab of plywood is embedded with 600 GOPs of compute capacity[1] and gets machined into a table top, either the table is going to become complex or the computing is going to become transparent — and with that many MIPs to spare, most of the lay public will gratefully opt for the transparency.

Focus

The focus of this research will be on the programming model for the *paintable*. Given that neither the hardware, nor the programming model, nor the applications yet exist, this choice may at first appear somewhat arbitrary. However, if one considers the challenges in actually realizing a paintable computer in the context of today’s technology, the largest leap of faith is, by far, the programming model.

Steady progress has brought the paintable’s hardware in sight, if not within reach...

Advances in integrated systems have been so regular over such a protracted period that the phrase Moore’s Law is long since a household word. Most of the hardware capabilities predicated by this thesis proposal are well aligned with the expected near term advances in IC process technology, with power harvesting being the notable exception. While not yet on the shelves at Fry’s[2] the hard-

1. 750 processors per square foot, each with a micro running at 50 MHz
2. one of the original consumer electronics superstores, located in Silicon Valley

ware is currently within a tractable number of engineering cleverness units away from being buildable. However the willingness to invest those cleverness units is gated on the appearance of a practical programming model.

By contrast, the art of decentralized programming for fine grain parallel machines has advanced in fits and starts with no dominant direction or consistent rate. Proposed techniques are often tightly bound to exotic hardware architectures whose only advantage over an expensive von Neuman contemporary is speed. Many of these programming techniques rise and fall with the hardware in a marketplace where the novelty of the programming model is a burden to those who would be obliged to an expensive retrofit of preexisting application software.

... while progress on parallel programming has been sporadic and fleeting.

For progress on the software front to be comparable to that of the hardware, we would have to be making rapid, steady, and tangible progress toward a general technique for chunking application code into arbitrarily small segments for distribution onto dense arrays of computing nodes. Yet, to date, there is no such programming model in sight. Nor is there likely to be. I can not see the day when applications like MS-Word will be efficiently realized as several thousand minute code fragments running concurrently on slow embedded micro's.*The trick is going to be to identify largely ignored yet useful applications which add value in the commodity computing context*.

Approach

The steady advances in the hardware, the lagging, sporadic progress in the programming model, and the resulting dearth of applications all suggest an approach for this research:

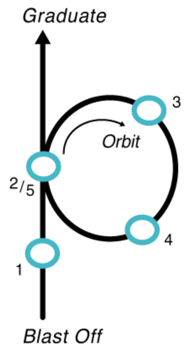
1) Define a hardware reference model for a single computing 'particle'.

2) Propose a system architecture consisting of models for the applications programming, the inter-particle communication and the external I/O.

3) Enshrine this system architecture into a simulator.

4) Develop representative applications.

5) Use results from applications development to drive a round of refinement on the system's architecture. Loop back to step 3.



The hardware reference model is a behavioral definition of a computing particle, with the details of the various sub-systems grounded in current engineering practice. Less a final blueprint for an IC, the goal is to capture the essential features of the *paintable* at the level of a single particle. The reference model would capture features like the amount of memory, the clock rate of the processor, the range and bandwidth of the communication sub-system and the total power required.

The iterative refinement would begin with a first cut at the programming model and associated models for the inter-particle data exchange and the external I/O. A simulator would be written around this model and used as a platform for developing applications. Results from the application development would be fed back into another round of refinements for the programming model.

Likely Results

The ultimate goal of this work is to help unlock the latent potential of a powerful yet largely ignored architecture. In the best case, this work will help avalanche a rethinking of how best to distribute processing in a sensory rich environment. Along the way, several useful by-products will necessarily emerge:

- The programming model itself
- A simulator
- Descriptions of the various algorithms for which a *paintable* is well suited
- Several novel applications with quantitative performance analyses
- A table-scale environment for demonstration and test

We expand on these contributions later in a dedicated section.

Why the Media Lab

This work resonates with the media lab on a number of levels. It continues in the tradition of several members of the faculty. It builds on work ongoing in multiple groups. It exploits material resources unique to the lab, both internal and externally through the sponsors. It contributes to several pursuits which have been core to the lab. And in its execution, it will adhere to the lab's charter of mixing the eclectic other-worldly with the practical.

Superficially, this work could be characterized as distributed computing meets miniaturized sensing, meets embedded networking, meets visual and audio scene analysis, meets societies of miniaturized mobile agents. As this proposal unfolds, I hope that the reader will sense the influence of several members of the faculty; notably Neil, Mitch, Pattie, Marvin, and of course, Mike and Andy. The overlap extends to several ML projects, both past and present. Examples include Rob Poor's work on embedded networking, the distributed computing efforts of Jacky Mallet and Nelson Minar, Bernd Schoner's work on physical modeling, Dan Ellis's thesis work on prediction driven analysis, the Star-Logo project in Mitchel's group, and the structured video work in the garden. All of these efforts have directly influenced the approach taken in this work.

To the extent that we can, we want to learn by building. And the greater Media Lab community is a unique concentration of many of the requisite resources.

- PhM for sensors and comm
- the IC design / fab resources of the sponsors
- the compute resources required to develop applications on large scale simulation.
- experience in room scale HCI (I/O bulb)

Topically, the notion of a paintable computer is not unique to the Media Arts and Sciences. Indeed, the epicenter for related work on the hardware and core system software is the CS departments of select universities. Premier among them is the amorphous computing group at MIT. A joint effort of the AI lab and the LCS, their work has done much to define the field. Nevertheless, this field is still a young one, the programming model still a question mark, and applications in both commerce or the arts have been notably lacking. The Media Arts and Sciences' unique offering to this field derives from its expertise in video, audio digital expression., and past work on cooperative systems.

Roadmap

The remainder of this document discusses the project in detail. The next two sections review the relevant background. Section 2 surveys recent hardware efforts for ultra dense computing, and positions the paintable computer in this context. Section 3 describes the challenges of constructing robust programming models for parallel machines, and looks at why past efforts have failed. Section 4 advances an alternate approach based on self-organizing behavior of miniature mobile agents.

The remaining sections contain the nuts-and-bolts of the proposal. Section 5 describes the proposed work at the level of general strategy and specific execution. Section 6 lists the expected contributions. Section 7 revisits the question of why this work should be done at MIT Media.Section 8 is the fine print — scheduling, required resources, and deliverables. The body of the proposal concludes with a fictitious bio and some unread references.

Two appendices and two separate documents are attached. As of this writing, approximately a third of the ML faculty have kindly offered their time to privately discuss this proposal in one of its formative stages. Appendix 1 catalogs some of the most ardent objections to this work - along with my equally ardent responses - into a list of *Frequently Flung Arrows*. The argument for the inevitability of a paintable-computing-like hardware quickly turned into an extended rant. The body of this rant has been broken out into a separate essay and buried in appendix 2. This need only be read by those who question the relentless progression toward this architecture.

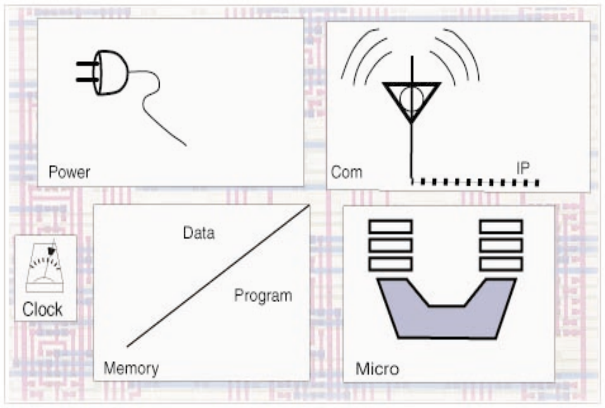
The attached documents are the data sheet and a survey article from the last chip I shepherded through an IC fab before entering the Ph.D. program. It is a mixed signal device with a full featured a base-band video decoder. The aggregate processing power is well in excess of a 386. In the IC geometries common in 1995 (0.8 micron) the actual die area was appr. 40 mm^2 — about 1/6 the size of a dime. In today's geometry (0.18 micron), the area would be under 3 mm^2 — about 1/5 the size of a match head. If manufactured in 0.1 micron, it could not be sold stand alone[1].

2 Paintable Hardware

This section outlines the hardware architecture of a *paintable* and positions it in the context of parallel computing. The notion of a paintable computer is distilled to a list of characteristics. These characteristics are enshrined in a first cut hardware reference model for a single *paintable* microchip. Ensembles of these chips collectively constitute a paintable computer as an instance of massive parallelism. Past work in parallel machines is surveyed in tabular form, and the *paintable* is compared to it's closest relatives.

The hardware reference model for a computing 'particle' consists of:

- Power harvesting Subsystem
- A wireless transceiver for local communications
- A general purpose micro processor
- Order 100k of RAM



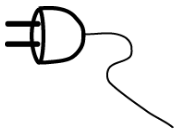
1. When the die area shrinks below a certain point, it is no longer economical to package them separately, unless the production volumes are immense.

Paintable Particle

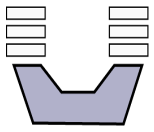
Up to this point, the notion of a paintable computer has been vaguely described as computational elements which can be painted onto an arbitrary surface and which somehow organize themselves to do useful work. Structuring this research requires a more formal description of the hardware. As an aid to definition, we propose a hardware reference model constructed around a single IC with dimensions 2 mm x 2 mm. Onboard subsystems include a block for power harvesting, a full featured micro, an RF transceiver for inter-particle communication, a 50 MHz internal clock, and appr. 50K RAM for program and data space. Each of these subsystems is described in more detail below. Less a production grade design, these specs are intended as a reference grounded in current engineering practice. One area where current practice fails is in the nomenclature. Throughout the remainder of this document, we adopt the name ‘particle’ to describe this IC.

Throughout, we refer to the individual computing nodes as ‘particles’.

The power harvesting subsection must couple to an external power source without requiring precision connections or placement. The chip should be able to assume any orientation and any position within a defined area and still receive some amount of power. To compensate for fluctuations due to the chip’s position relative to the source, the power subsystem will regulate the system clock to match the chips’s requirements to the available power. Several candidate techniques using capacitive or inductive coupling are available, but none of stock methods yet offer close to sufficient power. In terms of available cells from industrial libraries, this subsystem is the least mature.



By contrast, seasoned designs for a suitable micro-processor are comparatively plentiful. A rough cut at a standard spec would be a standard ‘08 class embedded controller, pulling 500 uA per MHz clock and occupying appr. 2 mm^2. A number of peripherals are also commonplace; FLASH storage, timers, UARTs, and low fidelity A/D ports.



The inter-particle communication would borrow from related work on IC’s for near-field RF. Much remains to be clarified regarding the details of the modulation technique, the bandwidth / power / distance trade-offs and methods for local selectivity. However, there is a good body of past practice from both research and industry to draw on. For the reference model, we assume a minimum bandwidth of at least 100 kb/s in both directions.

In typical operation, an ensemble of these particles would be deposited on a surface in close proximity to each other. Once exposed to power, each particle would boot, contact it’s neighboring particles and dynamically configure the ensemble’s address space.

The method for supplying code for the on-board micro is described below in the section on the programming model.

While ill-suited as a blueprint for product, this reference model does capture the requisite characteristics for a *paintable* particle:

1) no precision placement
2) no dedicated interconnects
3) no need to differentiate or sort particles by functionality
4) asynchronous timing model
5) networking model based on spatial locality
6) vanishingly small unit cost ($0.002 / MIP)[1]
7) node size on the order of a pin head

To expand on this last point. While the results from this research will likely be applicable to systems which operate on a much larger spatial scale (for example, that of a factory floor) we specifically target applications where several thousand processing nodes fall within an arms length of a human. While this delineation is admittedly arbitrary and will not be rigidly enforced, it does reflects a strong personal bias that this is where the interesting new applications lie.

Related Work

Fascination with parallel computing dates back at least to the appearance of the microprocessor. The following table lists some representative efforts and groups them by granularity, scale, architectural maturity and the degree to which computing is task-specific.

Architecturally, the *paintable* is most closely related to work on the micronscale machines. Both employ mature logic components based on stock silicon processes, hence the moniker ‘micronscale’. Both define systems as aggregates of miniature autonomous computing elements, running asynchronously, communicating locally and deployed by the hundreds or thousands.

In contrast to the *paintable*, most of the micronscale projects focus on advancing the monolithic integration of computing, networking and microsensing. Target applications are those which involve spacing between the particles on the order of one to ten meters (ex. inventory control, environmental temperature control, and battlefield surveillance). No general purpose programming model have been proposed and the architectures are necessarily task-specific.

1. Die cost of $16.00 / in^2 Die size of 4 mm^2 50 MHz clock speed

Parallel Computing at a Glance

Family	Research / Products	Defining Characteristics
Traditional Parallelism (coarse grain)	• shared memory multiprocessors • networked distributed applications	• process level parallelism supported by threading • commercially successful
	• VLIW	• instruction level and sequence level parallelism • data Flow Programming Model with control flow scripted by compiler • application specific
Traditional Parallelism (fine grain)	• Connection Machine • MassPar	• instruction and sequence level parallelism • overtaken by conventional architectures • unconventional programming models • expensive
Distributed Control Grain (Coarse) ↓ (Fine)	• cars • TV's • ubiquitous computing [20] • Smart Matter [4]	• process level parallelism • coarse grain variants already commercially successful • fixed functionality, not programmable
Distributed Sensing / Micronscale Computing	• BSAC [11] • WINS [3] • MTL (MIT) • Sensor Fusion	• still predominately a research effort • fine grain parallelism with node counts in the thousands • ad hoc, task-specific programming model for system level • order unity ratio of sensors/actuators to processing nodes • traditional logic components assembled in non-traditional system architectures
Nanoscale Computing	• cellular computing [1] • CAEN's [5] • quantum computing [9]	• early research phase. Still assembling simple gates • ultra-fine granularity • unconventional substrates (biological, molecular, atomic) • too early to worry about a programming model.

3 Programming Models

While the basic hardware architecture is a commonly shared ideal, the programming model appears to be a commonly shared question mark. In this context, the phrase “programming model” collectively refers to the rules for assembling the machine instructions into functions, the means by which the functions exchange data among themselves and with the external environment, the mechanisms for grouping functions into applications, the management of limited system resources (memory, CPU cycles), and the authoring environment.

This section considers the boundary conditions imposed on the programming model by the *paintable* architecture. We synthesize a set of basic requirements, compare these requirements against past work in parallel processing, and highlight the open problems. Self organization is advanced as an attractive alternative to the traditional approaches.

Boundary Condition from the Hardware

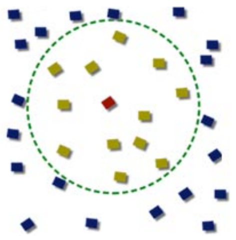
An instance of thousands of miniature processing nodes, running asynchronously (if at all) and communicating locally via an ad hoc network places unusual demands on the programming model. Worst among these are:

A checklist for paintable software:

- Asynchrony
- Fault Tolerance
- Network Locality
- Adaptive Topology
- Code Compactness
- Shared Data
- Mobility

Asynchrony Clock level synchrony is out of the question. 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 what processes will be running on a neighboring particle. Code running on one particle should never explicitly synchronize to events generated on another particle.

extreme **Fault Tolerance** Allied with the inherent asynchrony is the propensity of individual particles to fail completely. A defining characteristic of a paintable computer is that the user should be permitted certain tasks 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. While the size of the neighborhood can vary substantially, current experiments run on neighborhood sizes ranging from 8 to 20 particles.

Adaptive Topology Any truly *paintable* system will have final topology which is unknown at the time that much of the application code is written. While it will always be possible to recover an approximate coordinate system at run-time, no application code should rely on a particular spatial layout of the processors. As a consequence, no application code can explicitly address a processing node by location — neither as an absolute location nor as a relative location (ex. two hops north).

Code Compactness On-particle memory is very limited, inter-particle comm bandwidth is slow compared to processor speed, and there is no external support for virtual memory. Functions running on a given particle should therefore be self contained and sized to fit completely in a single particle.

Shared Data Nevertheless, the utility of a single particle's computation will often go up if it has access to results from local computations on neighboring particles. With the caveat that no process can predict what processes are running in the neighborhood, tagged data passed from the neighborhood should be available to processes running on a given particle.

Mobility Capacity for inter-particle migration of code segments will increase both the functionality of the individual particles and the adaptability of the overall system. The restriction here is that exact trajectory of the migrating code can not preordained.

Past work and Open Problems

This is a singularly challenging venue for any known programming model. And surprisingly, there has been little published work on suitable models from those people most in need of it — the research groups doing the ultra dense micronscale hardware. To date, the literature from these groups contains almost no explicit treatment of a programming model[1].The notable exception being the amorphous computing group at LCS, whose work is considered in some detail later in this section.

Conversely there is an ocean of prior art on programming for parallel systems. Examples include data flow[10], distributed computing[14][15], decentralized computing[8], and the custom operating systems for special purpose hardware[12]. The most relevant subset is the work on fine grain parallelism, where the capacity of the individual processing nodes are relatively modest and size of the ensemble number into the thousands. Programming models for these machines get us some of what we are after, but suffer a cru-

1. When I figure out what it is about this problem that they know and I don't, I am going to feel really silly.

cial limitation. To develop this point, we look one of the *paintable's* architectural predecessors, massive parallelism on the Connection Machine.

SIMD and the Connection Machine In a Single Instruction Multiple Data (SIMD) architecture, the machine's memory is divided up among multiple processors with each processor privately managing its share of the memory. A single instruction sequence is broadcast simultaneously to every processor which executes the instructions in lockstep. Each processor can optionally 'sit out' an instruction depending on its internal state. Interprocessor network topologies vary, but no node can commence with a particular instruction until all the nodes have completed the previous one.

An elegant exemplar of this architecture was the Connection Machine (CM) from the now defunct Thinking Machines Corp[12]. The CM 1 and 2 were prototypical instances of fine grained parallelism. At a full compliment, there were 64 K[1] bit slice processors,[2] each with approximately 1K bytes of private local storage. Interconnect topology was a 16 dimensional hypercube. In this address space, each processor node had 16 immediate neighbors. While any one processor could send messages to any of the others, the 16 'neighbors' in a sense constituted a locality with messages to any of the processors outside this locality requiring multiple hops. The central insight of the CM was that, *for a parallel machine to be useful, the topology of the machine had to be well matched to the natural topology of the problem*[3] and that by nature, the n-dimensional hypercube could efficiently emulate a wide variety of useful topologies. Much of the CM's development effort went into maximizing the speed and flexibility of the router — hence the name Connection Machine.

In operation, an external controller first mapped the topology of an application onto the 16 D hypercube as a preprocessing step. The resulting data structure was serially loaded into the CM and operated on in parallel. If the nature of the problem was such that operations performed at all the nodes of the data structure were similar or at least clustered into a small number of groups, SIMD worked well. If they were not, the SIMD machine was reduced to an inefficient emulator of a MIMD[4] machine.

1. K = 1024
2. Later versions employed powerful SPARC processors to approximate a MIMD architecture.
3. example: Parallel nodes arrayed on a 2D Cartesian grid is a natural topology for the processing of still images.
4. MIMD: Multiple Instruction, Multiple Data

So, how does the CM brand of the SIMD programming model rate on the *paintable*'s score card?

On the plus side, the CM was moderately fault tolerant, could support a locality restriction, could share data between nodes and could adapt to limited variations in the topology of the hardware. On the down side, the CM rigidly enforces lockstep synchrony at the instruction level. With the instruction stream supplied serially from an external controller, there was no motivation to produce compact code. Similarly, code migration as envisioned in a *paintable* runs counter to the basic definition of SIMD.

A *paintable* score card for SIMD:

	Asynchrony
xx	Fault Tolerance
xxx	Network Locality
x	Adaptive Topology
	Code Compactness
xxx	Shared Data
	Mobility

Not very promising.

What price Engineering Determinism? Given that the paintable is basically a MIMD machine, why dwell on the shortcomings of SIMD? Because SIMD on the CM lucidly illustrates a fundamental problem endemic to all models for parallel programming — engineering determinism. Consider the effort involved in building a programmable machine with 1000 two-input adders running in parallel. Adders are vanishingly cheap. And getting 1000 adders to concurrently sum their input is not hard. The magic is getting them to add the right two numbers, at precisely the right instant, and to pass the result to the right destination, all under control of a program written by a human. In massively parallel systems like the CM, the overhead for this control completely dominates the system cost. Our willingness to tolerate this 'control surcharge', however excessive, traces back to the belief that the only reliable order which can be foisted on a complex system is an order which is authored by a human.

Transparent systemization — the ability to proscribe the state behavior of a system down to the finest component level — is a design dogma born of engineering necessity. In an era when the individual processing elements were expensive, this 'necessity' made good sense and was seldom challenged. But as unit costs for the processing nodes tumble, the overhead costs of maintaining explicit control of an ensemble rapidly increase. The rate of this increase suggests that engineering determinism as a design strategy will fail to scale. For those who would build a *paintable*, this forces the search for an alternative.

Self Organization

Such an alternative has emerged in the form of self organizing systems. These are systems where useful organization and behavior emerge on a macro scale as the result of interactions confined to a much smaller micro scale. Small scale, local interactions — often completely defined by a few simple rules — result in large scale behavior that appears to be very complex — so complex that early efforts to model it usually assumed intricate centralized control.

Self organization is treated formally in the work on Complex Adaptive Systems. Local interactions operating randomly form the basis of distinct feed forward and dissipative mechanisms whose competition guides the overall system toward one of several equilibrium states. These equilibrium states can often manifest themselves as observable structures. Changes in the composition of the global environment alter the balance between the competing mechanisms, causing the decay of the existing structure in favor of a new one. Critically, the global behavior of the system derives from the balance of the competing processes operating on the macro level. And those processes in turn emanate from the rules governing the local interactions.

For engineers, this is both good news and bad news. The good news is that, because the global behavior depends on the *rules* defining the local interactions (as opposed to any specific instance of an interaction), the behavior is largely robust to structural variances. For example, the pressure in a gas vessel does not depend on any one of the molecular collisions. The bad news is that emergent behavior often can not be characterized at a sufficient level of detail. For example, while specific types of patterns can be reliably generated, there is limited capability to define the exact shape of the pattern.

Self Organization - Applications

An increasing number of applications are structured around self-organization

In nature, there are many systems that depend exclusively on self organization to manage their affairs. In engineered systems, there are a number of applications which employ self organization. However, almost all of these applications are implemented as conventional structured programs running on general purpose machines. Typical self organizing apps include Cellular Automata[19], Self Organizing Maps[13], Cluster Weighted Modeling[16], load balancing techniques for telecom networks[17], behavioral systems for synthetic characters (ALIVE, Synthetic Fish Bowl), colonies of ant-like robots for remote exploration[18], computing on graphical networks and Bayesian Belief propagation[7].

Two of these appear to be particularly relevant to a *paintable*.

CA Cellular Automata[19] is perhaps the most familiar instance of self organization at a scale similar to one targeted in this work. CA was described in the 1950's as a technique for simulating the behavior of fluids and gases. Computing nodes are arranged in a dense, regular lattice with a fixed number of neighbors. At the micro scale, nodes limit their communication to their immediate spatial neighbors, passing tokens which represent the arrival or departure of a gas particle and its properties. Simple update rules describe the outgoing trajectory of the visiting particles. At the macro level, the behavior of the gas conforms to global models described by differential equations.

CWM Cluster Weighted Modeling[16] is an example of the dimensionality reduction techniques used to generate models of complex physical systems. In CWM, sampled input and output are collected and used to train an ensemble of interacting clusters. Each cluster embodies a simple (often linear) transformation as a model for mapping between input and output. The micro level consists of the individual clusters. Each clusters employs EM to jointly compete for subsets of the input space and adjust the parameters of its transform to better approximate the output space. At the macro level, the weighted superposition of the cluster outputs define the response of the physical system.

Self Organization Architectures

While self organization appears often at the application level, there are only a few instances of whole systems being built from the ground up around self organization. A definitive example of such a system is the work of the amorphous computing group at AI/LCS[1]. They employ self organization in the strict sense; all computation is based on local interactions only. In their original programming model, all code is permanently embedded into 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 — including the arrival of message tokens passed from neighboring particles.

Self organization is almost never found as a core element of an operating system.

A growing point language (GPL) has been developed for approximating planar graphs onto a single 2D layer of particles[6]. Communication of state data between the particles is implicit in the diffusion of 'pheromones'. At each particle, the local pheromone profile is sensed and used to compute a tropism[1] which directs the propagation/replication of migrating 'growing points'. On entry into a particle, the growing point 'activates' elements of the particle's code which analyzes the particle's local state and performs any or all of three tasks:

1) produce pheromones which are secreted to neighboring particles.

2) produces/updates static local symbols intended to represent predefined material properties.

1. This work relies heavily on biological metaphors. A 'tropism' in this context can be regarded as a function of several gradient fields

3) directs the further migration of the growing point.

Those who have done too much UNIX programming could regard the particles as an array of locally interconnected computers and the growing points as wandering interrupts, embodied as tokens. A local copy of all the code necessary to service the interrupt resides on all the machines. In the absence of an interrupt token, the machines communicate locally by passing a small number of predefined housekeeping messages back and forth. When an interrupt token arrives, the service routine wakes up and selects internal functions based on the state of the machine and the parameters passed along with the token. On completion, the token is passed on to one or more of the neighboring machines.

Given a distribution of particles on a regular lattice, any planar graph can be expressed as set of GPL commands compiled down to a set of local commands for embedding into all the particles. Useful performance has also been demonstrated when the requirement for distribution on a regular lattice is relaxed.

4 Programming a Paintable

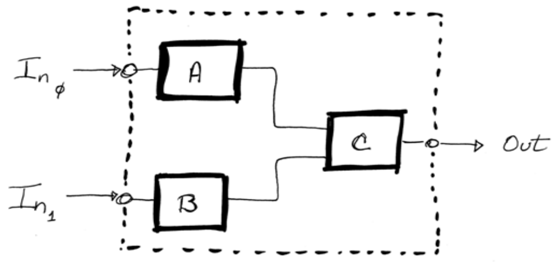
The research proposed herein would apply self organization to construct a processing environment where application level software modules self assemble from randomly distributed code fragments. The self assembly would use sensor input as a driver and the constituency of the code fragments as a guiding boundary condition.

Self Assembling Code

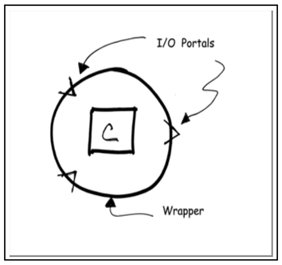
To contrast this with conventional programming practice, consider a simple function consisting of three interconnected subroutines. Conventional programming techniques proscribe a priori a fixed set of predefined paths for the data flow between the subroutines. Metaphorically, one could regard the subroutines as blocks positioned in a static scaffolding.

Tradition:

Program flow is encased in a fixed structure with predefined branches.

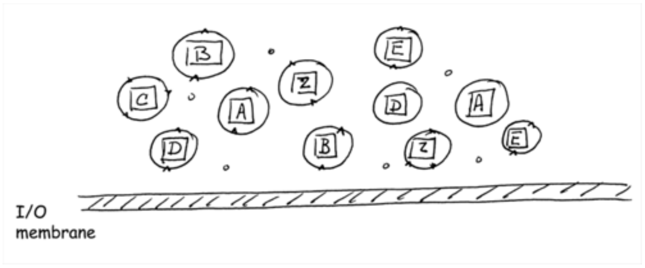


Functions which self assemble follow the alternative metaphor of organisms swimming around a medium, colliding randomly and interacting in response to chemical signatures embedded on their surface. In practice, we would approximate this by considering the machine's memory space as a fluid medium in which tagged data can be arbitrarily positioned. The individual subroutines would be encapsulated in an active wrapper that would support mobility, couple to the wrappers of other subroutines, and interact with the tagged data.



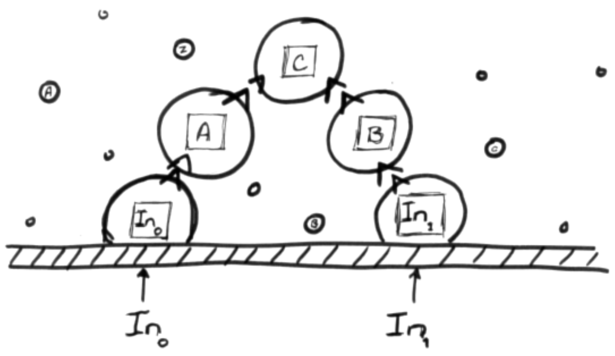
Alternative:

Individual subroutines are enveloped in a 'wrapper' which supplies mobility and gates the I/O access.



Mobile code fragments diffuse randomly throughout the system memory.

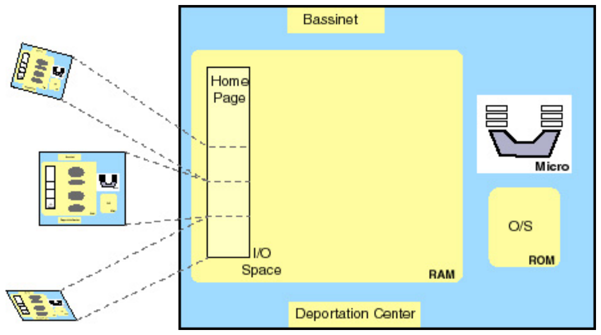
In the absence of input data, the dissipative mechanism dominates, suppressing the grouping of the code fragment, leaving them to diffuse randomly throughout the memory. The arrival of the relevant input changes the balance, naturally fostering certain groupings of the code fragments. This self assembly would continue until the feed forward and dissipative mechanisms arrived at a new balance, supporting a stable grouping of the code fragments into a macro function.



Input data appearing at the memory's 'membrane' is enveloped in a wrapper' and seeks to catalyze a self-organizing structure.

Fanciful? Perhaps. Yet, a preliminary definition of the programming model for the *paintable* has been developed which captures much of this dynamic. This model is outlined below in three parts; the organization of the RAM space, a normative definition of the code fragments and a description of how the two interact.

Memory Allocation



organization of the RAM space of a particle

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

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

The I/O space could be regarded as a public bulletin board, where the HomePage portion is writable and the entire I/O space is readable

Code Segments

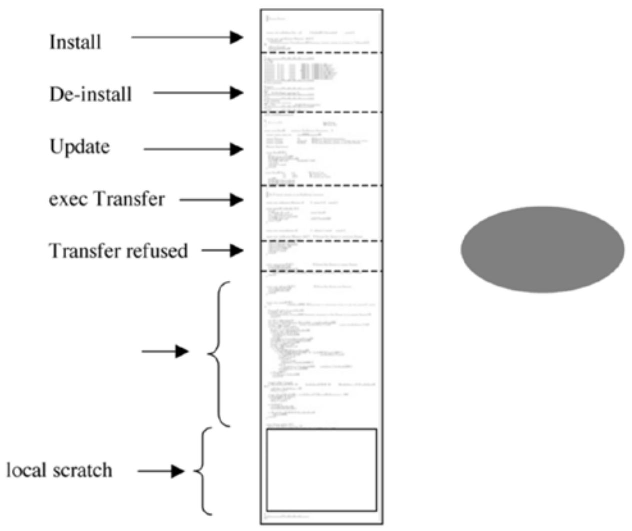
All software intended to run on a particle's micro must be organized into autonomous modules — or code segments — which conform to three normative requirements:

1) They are self contained modules capable of fitting entirely in the RAM space of a single particle. Here, the phrase 'self con-

tained' means that they do not explicitly depend on subroutines or functions which are external to the code segment.

2) They gate their entire I/O through the I/O space in the particle's RAM.

3) They define some support for 5 simple functions which the particle's O/S can give them. These five functions are *Install*, *De-Install*, *Update*, *Transfer-Refused*, and *Transfer-Granted*.



Code Segments are self contained functions which:

- fit completely in the RAM space of a single particle
- have handles for at least 5 commands.
- gates all I/O through the particles I/O space

A legal, if near useless, code segment would be one that answers a call to *Install* by posting a "Hello World" to the HomePage, answers the *Update* command by posting the text "I'm still here," answers the *Transfer-Refused* command by posting "Lonely Heart seeks Soulmate," responds to the *Transfer-Granted* command by erasing it's posts and migrating to a neighboring particle, and responds to *De-Install* by erasing all of it's posts and erasing itself.

RAM space permitting, a particle will accommodate multiple code segments simultaneously. But it is up to the particle's OS to enforce any pre-defined boundaries on RAM usage.

Run-time Scheduling

At run-time, code segments migrate nomadicly looking for particles on which to install themselves In those particles where entry into the program RAM is successful, the code segments will set up shop and begin searching for relevant data in the I/O space. The side effect of the code segment's activities its additional posts to the HomePage. Often, the number of code segments seeking entry will exceed the particle's capacity. The allocation of program space is regulated by the O/S in response to competition among the code

Behavioral description of Programming Model:

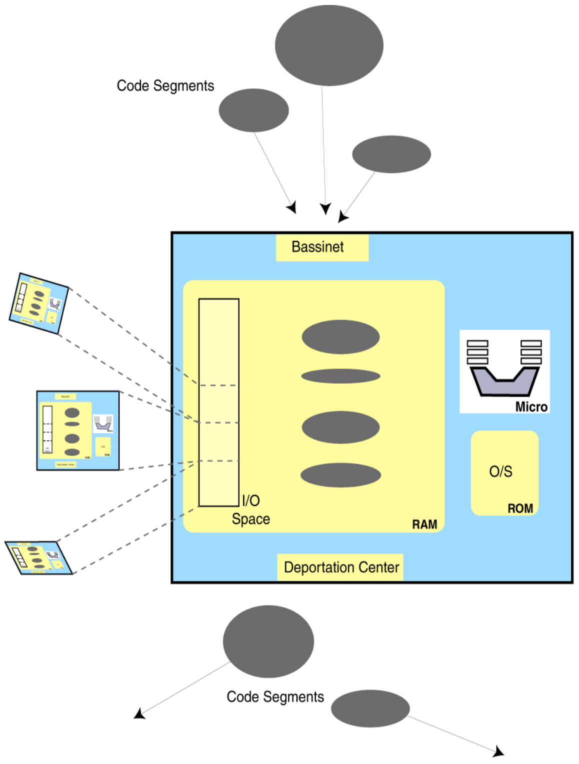
Code segments wander nomadicly between particles,

.. pass through the active FIFO {bassinet) and are installed in the RAM space

... communicate via tagged posts to the I/O page

... compete for available space in the program RAM

... with the losers queued for deportation to the neighboring particles.



segments. Each code segment must draw it's competitive advantage from the I/O space and therefore, indirectly from the activity of other code segments. The competition is arbitrated by the particle's OS. And when a particular code segment loses out, it is de-installed and passed to the output port to migrate further via diffusion.

Metaphorically, the contents of the I/O space can be compared to soil with a particular nutritional profile. The code segments are in turn comparable with plant life trying to take root in the soil while concurrently contributing to the soil's nutritional capacity — albeit without depleting the existing storage.

5 Proposed Research

For those readers who have taken the short cut to this section, the starting point for this thesis work can be summarized as follows:

> The ability to manufacture autonomous computing nodes on the scale of sand kernels can now be regarded as preordained. The progress of sensors and actuators is following a similar curve, albeit with some delay. Ensembles of these processors / sensor particles will be embedable into to ordinary surfaces at densities as high as a thousand per square foot. A programming model which requires that

each of these particles be individually programmed and monitored will exceed most people's threshold for managing complexity. An alternate programming model built around self-organization would be an attractive alternative.

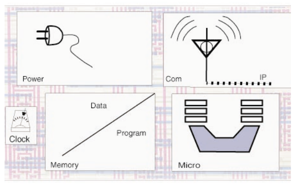
Strategy

The thesis work will begin with a tractable hardware reference model and a set of criteria that the system must satisfy. This will produce boundary conditions on the software that we will accept as immutable. A programming model will then be proposed and iteratively refined, with each iteration driven by insights from the development of several applications in a simulated environment. In creating these applications, the challenge will be to reduce some desired global behavior to a collection of unreliable, poorly synchronized local interactions operating on the scale of a single communication radius. As work proceeds, the changes to the programming model should become less and less constitutive. Ultimately we do not expect to arrive at the demonstrably best programming model. Rather, the intent is to identify a model with merit, demonstrate its efficacy and explore its limits.

Execution

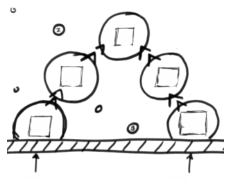
Work will proceed in five interdependent threads, with each thread feeding results back on the ones before it. These five threads are the Hardware Reference Model, the System Architecture, the Simulator, sample Applications, and breadboard emulators for the actual hardware. These threads are interdependent in obvious ways. For example, before work on any of the other four threads can begin, an provisional hardware reference model must be built around some reasonable assumptions regarding the particles basic properties. Yet, work on the simulated applications and the hardware breadboarding should highlight the need for some changes and/or additions in the reference model.

Hardware Reference Model The purpose of the hardware reference model (HWRM) is to establish reasonable bounds on the capabilities of the particles. In particular, a detailed model for the wireless subsystem and the resulting interconnect topology will have to be established. The HWRM will be defined with one eye on current engineering practice and another on the core characteristics of a paintable computer.



An example a of compromise which can be made at this level involve the details of the power subsystem. It is not clear that contactless techniques for power transmission will supply sufficient power for the particles to do anything interesting. Yet, the core requirements of a *paintable* could in fact be met by an alternative involving direct supply of power, provided that the restriction against precision placement of the particles is upheld.

System Architecture The system architecture consists of the programming model and the associated models for external I/O and inter-particle communication. This work forms the heart of the dissertation, and will undergo constant refinement throughout the course of the work.



To appreciate the challenge taken up in this work, consider the nature of the hardware. On the one hand, 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, the network message flow is chaotic and the unit reliability is low. On the other hand, this same hardware sports enormous raw compute capacity, vanishingly small unit costs and extreme ease of handling. Our ability to tap this potential turns on the programming model's ability to overcome the obstacles.

As a solution, I propose employing self organization at the level of the code segments — autonomous software fragments which embody some atomic function (ex. simple integration) and employ several strategies for executing that function in a dynamic, randomly changing environment. Code segments would compete for access to system resources (memory / CPU cycles) in a computing venue designed to permit each code segments to selectively aid or impede the others. Based on the competition, the code segments would organize themselves into macro functions in a manner reminiscent of the emergent behavior of adaptive systems.

Simulator The simulator fulfils several important needs. The first is visualization. Development, debugging and performance analysis invariably involve observing snap shots of selected state data for large groups of particles. Assembling these snap shots serially by querying each of several hardware particles would be onerous and error-prone. Software control of the visualization will often be the only viable option.

Simulation also offers an escape from the "catch-22" of manufacture. Sans a simulator, the applications could only be developed on running hardware — hardware which most manufactures would decline to build in the absence of some detailed description of the target apps. While bread boarding may be an option, the changes occurring early in the development cycle are likely to be too frequent and too elemental to practically realize in hardware.

The simulator will be one of the key deliverables of this research. An initial skeleton version, modeled after the Amorphous Computing

Group's HLSIM[2] is currently running. It models the particles behaviorally and suffices for the development of simple applications. It is written in Java to support portability and ease of threading[1]. As the research progresses, the software will be constantly refined. Several of these refinements can already be anticipated:

- port to a dual or quad
- more realistic network model
- support for arrays of I/O portals with a density on par with the particles
- support for multiple layers of paint

Applications The applications qualify the programming model and serve as a general sanity check on the concept of a *paintable*. In the course of this work, I am envisioning the development of four — still to be determined — applications. Each application will be selected with several criteria in mind:

Applications will qualify the programming model.

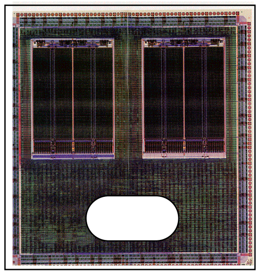
- it must demonstrate a compelling functionality, preferably a compute intensive one
- it must showcase a particular class of algorithms (e.g., modified EM)
- it should guide the evolution of the simulator
- serve as a vehicle for refining the programming model

Each application will be written up to the level of an internal technical report and, where reasonable, a webpage with a demo will also be provided. The four algorithms and associated applications are listed below (as Algorithm / Application) This list is tentative and subject to change as the work progresses. However, while still in need of much work, the first two are far enough along to demonstrate the concept. The last one (TBD) is the most ill-defined. The general intent is to apply a distributed technique for physical modeling in a feedback control app. In this case, the density of the sensor and/or actuators would be on par with that of the particles. As a preferred placeholder, a candidate app would be acoustic room modeling for blind source separation.

Algorithm	/ *Application*
Diffusion	/ *Holistic Data Storage*
Patterning	/ *Surface Bus*
Statistical Inference	/ *Image Segmentation*

1. I am already sorry I said that

Cluster Weighted Modeling / (*TBD*)



?

Hardware Ultimately, the goal is to fab particles and build a system. But progress depends on commitment from one or more of the sponsors. In the face of this uncertainty, we are forced to rely on bread boards of discrete components. Within the scope of this thesis work, I will structure the hardware activities to serve two ends. Primarily, hardware development should qualify and support assumptions made in development of the simulator. Secondarily, the hardware development should highlight the comparative simplicity of the an IC and attract the attention of a potential provider of foundry services.

Concretely, I foresee at least one experiment involving board development; a proof of concept for an electrostatic alternative to the RF subsystem for wireless communication. In this alternative, the particles would still communicate wirelessly, but via electostatic loading of a resistive sheet as opposed to radio frequency wave propagation. This work will provide a baseline on the communication bandwidth between the particles. If successful, this experiment would also move the complexity of the chip design from ‘doable’ to ‘very easy’.

6 Contributions

Up Front

Superficially, this dissertation will proffer a novel model for the programming and networking of an ultra-dense array of loosely connected computing nodes. The model will be qualified with performance data from selected applications coded and tested on a simulator. These applications will be both interesting in their own right and representative of a broader class of algorithms.

A contribution more lasting than quickly obsoleted performance figures would be 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 will argue 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 an interesting set of algorithms can demonstrate compelling behavior with acceptable reliability, this dissertation will lend credence to the use of self organization as an engineering tool.

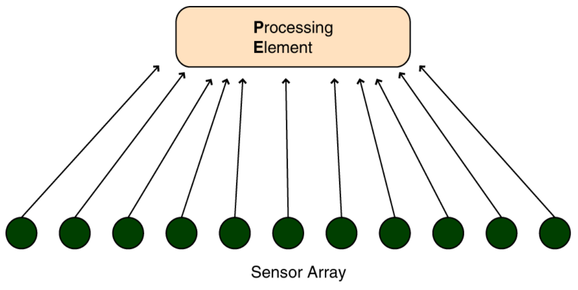
Cumulatively, these results should be an important resource for the system designer who is both attracted to the affordances of a *paintable*, and fearful of the limited application domain. All of this alone should add up to “*an original and significant contribution to* *knowledge*." It will even be a useful one if any of these applications really works.

Deeper Questions

On a more profound level, this work should produce insights into two deeper questions.

1) *In a dense, sensor rich environment, what is the appropriate degree of centralization for the processing*?

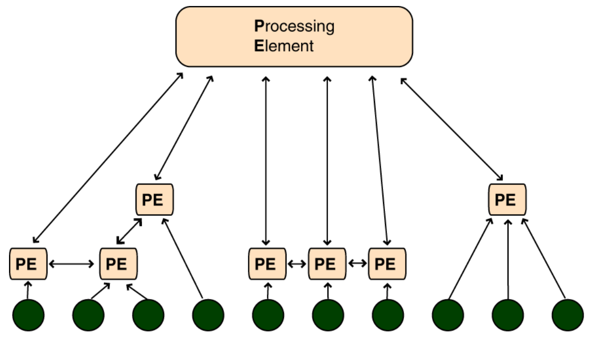
Traditionally, raw sensor data is shipped with minimal preprocessing to a central computing node for processing. This architecture was less a reflection of the basic nature of the problem as it was a bow to the relative costs of the computation, sensing and comm bandwidth.



Originally,

multisensor systems fed the sensor data directly to a central processing node.

The falling costs of dense sensor arrays and miniaturized computing nodes have driven alternative architectures where substantial processing takes place in the immediate spatial vicinity of small subgroups of sensing elements. An illustrative example of this trend is the work on artificial retinas, a single monolithic device where small groups of processing elements are uniformly interspersed among the elements of a photosensor array. Local filtering operations at multiple scales emulate the preprocessing thought to occur in the human retina.



Recently,

Falling costs and increased mobility have enable hierarchical architectures better matched to the particular problem.

In the design of practical computing systems, the architecture evolves in response to a number of factors: total system cost, sensitivity to unit failure, responsiveness to variable input behavior, cost of communication bandwidth, development time, and prevailing prejudices within the design community. For a given application, the complex question of the optimal cost must ultimately be answered in the marketplace. However by raising the bar on the mobility and granularity of the processing, this research could enable systems which are both adaptive and better suited to difficult problems.

2) *What is the nature of scheduling, multitasking, queuing and load management in a computing architecture that does not explicitly support any of them?*

The concepts of multitasking, scheduling, queuing and load management are essential to most computing environments and are explicitly supported in contemporary architectures. The programming model advanced in this work is characterized by miniature mobile agents, nomadicly wandering between particles, and competing with each other for processing resources. The basis for the competition is the input data available at a given location and the side effects of the operations of other nearby agents. Here, each agent is free to choose its own strategy for migration and competition.

In distributed, data-driven environments such as this, the overhead required to explicitly schedule and direct the agents is prohibitive. To the degree to which they are supported at all, scheduling, queuing, multitasking and load balancing will have to be implicit in the rules governing the agent's competition. Should this strategy compare favorably to conventional techniques (assuming it works at all), it would constitute an important result.

7 Why the Media Arts and Sciences

The Media Arts and Sciences is about communication and expression. And communication is about creating and exchanging symbols. The more content rich and expressive the symbols, the greater their potential for enhancing communication.

But the best symbols are useless if they are not easily accessible for a human. And much of the work at MAS bumps up against two basic limits: 1) the restrictive access to the internal state of a computer/network and 2) the architectural limits on the quantity and variety representations that can be built on the available data. In other words, the machine is not doing enough and we don't have a good view of what it is doing.

This thesis work explores an architecture that not only creates an immense amount of compute capacity, but that also draws it one step closer into the human sphere. Almost any environmental artifact can be given some amount of computing ability. Physical contact between objects not only supports communication between them, but integration of the distinct devices into a single computing whole.

The machine itself is not a MAS thesis, the MAS thesis is what we do with it. The architecture as proposed is, in effect, a massively powerful 'representation generator'. The computing strategy is to create a massive number of wildly differing representations and to structure their interaction such that, for a given problem, the most relevant representations “float to the top” into view. How we employ this strategy to produce symbols that enhance communication is the portion of the work which seems ideally suited for a Media Lab.

8 Fine Print

Deliverables

1) The simulator and supporting doc's
2) A table-scale demo running on the IO bulb
3) Tech report write ups for the four applications

Resources

For the table scale demo:

- a dual or quad multiprocessor (preferably a quad) with a mature Java 1.2 VM and support for video capture and display
- an IO-bulb installation, with a higher resolution projector.
- several software development environments (Java, C++)

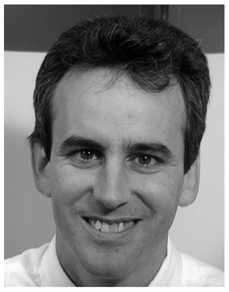
For hardware evaluation:

- parts and PC-fab services for the construction of at least 200 'motes' (emulator boards)

Time Line

July 3	Provisional hardware reference model	
	Tentative System Architecture	
	Spec down version of Simulator	
Sept. 4	Application	Patterning / Surface Bus
	Simulator	Port to multiprocessor machine
	Hardware	Testbed for inter-particle comm
Nov. 6	Application	Diffusion / Holistic Data Storage
	Simulator	Table scale demo
	System Arch	Tech report level write up
Dec. 4	Application	Inference / Image Segmentation
	Simulator	Preliminary Docs
	Hardware	Breadboard for networking particles/
Jan. 8	Simulator	Final Docs
		Updated networking model
	Dissertation	Outline
Feb 5.	Application	Physical Modeling / TBD
	Simulator	Support for dense IO arrays
Mar.5		Draft Dissertation
(mid Mar.)		Thesis Defense

Bio



A native of Washington, D.C., Bill received his SB and SM degree from MIT in ‘82 and ‘88 respectively.

In 1982 he joined the R & D department of ITT Germany in Stuttgart where he worked on video coding schemes for broadband ISDN. In September of 1986 he joined the Movies program at MIT’s Media Lab as a research assistant and basketball coach. From 1988 through 1994, he was a system’s designer / project manager in the Concept Engineering Department at Intermetall in Freiburg, Germany where he developed digital video components for consumer electronics market. In 1995, he joined MIT’s Media Lab as a research assistant where he works on programming models and algorithms for dense, decentralized computing ensembles.

An early participant in MPEG, he has authored several articles and holds 5 patents in the field of digital formats for video compression and storage. His interests include architectures for parallel processing, image coding and machine vision.

Preliminary Bibliography

[1] Hal Abelson, Don. Allen, Daniel Coore, Chris Hanson, George Homsy, Tom Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Embedding the internet: amorphous computing. *Communications of the ACM* 43(5), 2000

[2] Stephen Adams. A high level simulator for gunk. Technical report, Massachusetts Institute of Technology AI Lab, November 1997. http://www.swiss.ai.mit.edu/projects/amorphous/#hlsim

[3] G. Asada, M. Dong, F. Lin, T. S. Newberg, G. Pottie, and W. J Kaiser. Wireless integrated networked sensors: Low power systems on a chip. In *Proceedings of the 1998 European Solid State Circuits Conference*, http://www.janet.ucla.edu/WINS/biblio.htm

[4] A. Berlin and K. Gabriel. Distributed mems: New challenges for computation. *IEEE Computational Science and Engineering Magazine*, March 1997.

[5] C. P. Collier, E. W. Wong, M. Belohradsk, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, pages 391-394, July 1999. http://www.sciencemag.org/content/vol285/issue5426/.

[6] Daniel N. Coore. *Botanical Computing: A Developmental Approach to Generating Interconnect Topologies on an Amorphous Computer*. PhD thesis, Massachusetts Institute of Technology AI Lab, February 1999

[7] Robert Cowell Introduction to inference for bayesian networks In Michael Jordan, editor, *Learning in Graphical Models*. MIT Press, Cambridge, Massachusetts, 1999

[8] Robert Engelmore and Tony Morgan. *Blackboard Systems*. Adison Wesley, 1988.

[9] Neil Gershenfeld and Isaac L. Chuang. Quantum computing with molecules. *Scientific American*, pages 66-71, June 1998.

[10] Guang R. Gao, Lubomir Bic, and Jean-Luc Gaudiot, editors. *Advanced Topics in Dataflow Computing and Multithreading*. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[11] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: Mobile networking for smart dust. *ACM/IEEE Intl. Conf. on Mobile Computing and Networking (MobiCom 99)*}, Seattle, Washington, August 1999. http://robotics.eecs.berkeley.edu/~pister/SmartDust/.

[12] W. Daniel Hillis. *The Connection Machine*. MIT Press, Cambridge, Massachusetts, 1985

[13] Teuvo Kohonen *Self-Organization and Associative Memory* Springer-Verlag, Berlin, 1988

[14] Leslie Lamport and Nancy Lynch. Distributed computing: Models and methods. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science* Elsevier Science Publishers B.V., Amsterdam, 1990.

[15] Nancy Lynch *Distributed Algorithms* Morgan Kauffman Publishers, San Francisco, 1994.

[16] Bernd Schoner and Neil Gershenfeld *Cluster-Weighted Modeling: Probabilistic Time Series Prediction, Characterization and Synthesis*, chapter~15, pages 365-385. Birkhaeuser, Boston, 2000. http://www.media.mit.edu/~schoner/#publications.

[17] Ruud Schoonderwoerd, Owen Holland, Janet Bruten, and Leon Rothkrantz Ant-based load balancing in telecommunication networks. *Journal of Adaptive Behavior*, 5(2), 1996.

[18] Luc Seels. Cooperation between distributed agents through self-organization. In Yves Demazeau and Jean-Pierre Muller, editors, *Decentralized A.I.*, pages 175-196. Elservier Science Publishers B.V., 1990.

[19] Tommaso Toffoli. *Cellular automata machines: a new environment for modeling*. MIT Press, Cambridge, Massachusetts, 1987.

[20] Mark Weiser and John~Seely Brown. The coming age of calm technology. Web Document, Xerox PARC, October 5 1996. http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm

A1 Frequently Flung Arrows

In conversation with Media Lab faculty, several objections to the proposed work come up regularly. Here are some frequently flung arrows and my equally frequent responses.

Arrow

Determined attempts at parallel processing systems have come and gone, invariably yielding to advances in classic von Neuman machines. Today, there are well reasoned arguments that parallel computation will never be as efficient as von Neuman architectures. What do you know that they don't?

The question of efficiency continues to be debated. But the best answer to this question is to note its increasing irrelevance. As the price per unit MIP continues to fall, the question of how to use them to maximum efficiency becomes increasingly moot. With time, compute capacity will become a subordinate part of the design equation, being eclipsed by items such as ease of use.

Arrow

What will keep this from going down the same road of the Connection Machine?

Find me a Connection Machine that you can buy at Home Depot, paint onto a table top, upgrade with a paintbrush, 'pour' code into, and that continues to work with a 40% defect rate, — and I'll concede the point.

I believe that computation's diffusion into the environment will ultimately progress past the point of no return. In the case of embedded micro-controllers running distributed control, we are long since there. And it was economics, not fiat which drove this choice. Machines that embed several hundred GOPs into environmental mainstays such as furniture are operating on a different economic plane than the CM — even if those GOPs are not general purpose von Neuman GOPs.

It remains to be seen if this architecture goes anywhere. But the CM is no guide.

Arrow

IC design is risky and difficult to manage. We already have an impressive collection of networked general purpose machines. Can not much of this work be simulated.?

From my current perspective, I do not see any real hope of simulating several thousand computing particles on a collection of networked machines. The problem is not the compute capacity, it is the difference between the way processing gets distributed over a traditional network and how it gets distributed locally amid a dense particle ensemble. This is in part a question of my skill as a programmer and in part a matter of basic architectural limitations.

I see more promise in the use of a multiprocessor / shared memory machine; either a quad G4 from Apple or a quad Alpha from Compact. The simulator I am writing will be readily portable to such a machine and will likely take good advantage of the additional processors. This will support at least some simple examples.

But even if the bulk of the experiments are developed and run on a simulator, we will need at least some hardware development to qualify the decisions made in the design of the simulator and to scale up to really meaningful applications.

Arrow

Making hardware for hardware's sake is coming at the research from the wrong end. Constructing the hardware first and then searching for applications is a proven dead end — and you're not even promising to complete the hardware!

True, this work is unabashedly hardware driven. But the argument in favor of this hardware goes beyond the traditional arguments of increased compute capacity and the enabling of some exotic high end application. Unlike traditional sojourns into parallel computing, this proposal follows a relentless trend that is almost as old as the transistor itself; the trend toward cheaper, ever more powerful and accessible computing.

The economics of a *paintable* can be as interesting as its engineering. A function or program which runs on a *paintable* does not have to work twice as well as it does on a PC — it does not even have to work half as well — because if done correctly, computing which is buried in the table top could be a lot easier to use that the MIP's caged in the beige boxes. And a *paintable* only has to do a few things well to be worth the marginal cost.

Parenthetically, the argument against the "hardware first" approach ignores the history of the microprocessor. Briefly, Intel built the first micro expecting the mainframe and min-computer houses to snap them up. Ooops. Turns out the monolithic processor denied the target customers their value add. The calculator market was a respectable lifeboat. But the micro seemed destined for the economic sidelines until it grew its own market by enabling the development of the PC.

Finally, as of this writing, I have no idea if a killer app exists. But I explicitly reject the precondition on research that we exclusively follow the path of

problem -> algorithm -> hardware.

The instance of a novel, potent architecture in need of a single enabling advance is as valid a basis for research as any to be found.

Arrow

This is at least two theses! Getting this hardware to breath at all seems like a thesis unto itself. Thoughtful development of a programming model and some useful applications seems like another full time task.

The accent of this work will be on the programming model. The hardware will be treated as an adjunct... if perhaps a somewhat demanding one.

If treating design of an IC as an “adjunct” sounds crazy, consider the following. In a 10 month period shortly before entering graduate school, I “built” a commercially viable IC from start to finish (i.e. from paper to wafer) The crucial point is recognizing that the word “built” means doing the system design in the context of an industrial IC manufacturing operation. Given the requisite corporate support (circuit designers, layout, fab time, application support), the position of the project manager/ systems designer is one of those positions in the development food chain where the minimal effort produces the biggest bang.

This strategy obviously presumes the enthusiastic cooperation of at least one IC manufacturer. The necessary level of corporate commitment involves forces over which I obviously have limited control. Hence my unwillingness to unilaterally promise hardware.

Arrow

What evidence do you have that anything useful will run on this architecture? You appear to be starting from a point where there is little precedent to suggest the kind of applications which are well suited for your paintable.

This was perhaps the most difficult hurdle to overcome. I estimate that I have already done over a third of the thesis work just to produce two simple “Hello World” level applications; Holistic Data

Storage and the Surface Bus. While neither of these is compelling, they are both ‘real’ in that they are novel, serve a useful purpose and have the potential for finding their way into actual practice.

There is an argument which goes beyond the level of specific applications. It states that as the scale of the computing nodes shrink, relative to the objects of the ambient physical environment, the science of programming begins to look like physics. Namely, that the overall behavior of a system will be expressible by macro-rules in the form of `F=ma`, which are independent of specific computational events. This is not to imply that the laws of Newtonian physics will map directly. Rather it means that global computational behavior will emerge which can be modeled by tools such as differential equations.

None of this qualifies as evidence that the applications for a *paintable* are out there. But this does put this research in a sparsely explored regime of computing. We will have to see if anything comes of it.

A2 The Tide

This appendix outlines the rationale for the novel form of the proposed hardware. Quantifiable trends in process technology, economics and HCI are reviewed against the historical development of commercial and personal computing. We argue that the confluence of these trends set the stage for the rise of a dense, decentralized form of personal computing, which we later dub “commodity computing”. This discussion indulges in a measure of guesswork tempered by years of experience as an IC system’s designer for the consumer sector.

Computing’s Natural Modes

Already today, technology can support a near infinite variety of computing machines. Why then has only a small number of machine types emerged and prevailed? One answer is that computing is, in essence, a complex system. When people structure their computing, they do so in response to a myriad of interrelated forces:

Commercial computing is a complex system with natural modes... and paintable computing is a newly emerging mode.

- economics of development, production, sales, and distribution
- human affordances
- social conventions
- advances in allied technologies (display, sensing, software design)
- networked externalities (Metcalf’s Law)
- relative importance of the problem space
- alternative (low tech) solutions

Like all complex systems, computing organizes itself into natural modes, with each mode supporting a locally optimal trade-off between functionality and cost. This thesis work is predicated on the notion that *paintable* computing is on the verge of coming into its own as a natural mode. In this section, we characterize this mode by examining the forces which are shaping it.

Process Technology: The Driver

Few industries have a breakneck pace of change buried as deeply in their blood as the microchip industry. And the motor for this dynamism is process technology. Intel co-founder Gordon Moore gave us the coffee table maxim for this dynamic with his edict that the number of transistors will double every 18 months. But the regular halving of the transistor area belies the more powerful driver — the regular subsumption of previously disparate functions onto single monolithic dies. High volume, stock IC processes, which used to

Process technology is the driver forcing the diffusion of computation into the everyday environment.

restrict themselves to circuits for digital logic, now combine discrete logic, dense memory, and integrated analog subsystems for high speed A/D conversion, sensing and communications. A compelling contemporary example are recently introduced CMOS IC's which combine digital logic and optical sensing to yield single chip cameras.

This preordained rate of technical advance has, in turn, been enshrined in the economics of the IC industry, creating a textbook virtuous circle[1]. For IC's sold in large volumes, the first order determinant of price has always been the area of the die[2]. With every 2x shrink, silicon manufactures must sell 8x the functionality just to keep the revenues flat.

But flat revenues are not enough. Each succeeding generation of IC's typically requires either a new fab, or extensive remakes of existing plant, effectively raising the bar on the revenues. Additional momentum comes from manufactures of allied technology (displays, storage, software, communication, power), who build their product plans around the expectation of expanding functionality and / or falling costs.

In this environment, IC manufactures have evolved two preferred recipes for economic survival; 1) subsuming existing functionality within existing systems, and 2) pioneering new markets. Computing systems are typically assembled from IC's from various manufactures. With each shrink, the IC manufacturers must try to subsume as much of the total system as possible onto their die — ultimately yielding the system-on-a-chip offerings that have become common.

Even the largest of the existing markets are bounded. For regular access to the huge capital invest, most manufactures must recourse to the two axioms which have guided the industry since its inception: the genuine growth is in the new markets, and for chips which are regularly doubling in power, there are always new markets.

The IC manufacturer's creed consists of three laws:

- Expensive innovation is their meal ticket.
- New markets are where the money is.
- The ambient environment is the endless frontier.

All this adds up to an industry where the impulse for technical innovation has fed on itself to become self perpetuating, taking on a sense of urgency along the way. Many manufacturers have adopted as gospel the need to expand into virgin markets, with an obvious strategy being the steady encroachment into the ambient environment. The vehicle for this encroachment is IC's which are ever more autonomous and environmentally aware.

1. albeit one whose 'virtue' is apparent only to those who can keep up
2. The figure used throughout this document is $16.00 per square inch for a stock CMOS process

As a sanity check, consider the migration of computers from the air conditioned machine room, to the space next to the lab bench, onto the office desktop, onto the lap and into the pocket. With each lurch, the technology of computing overcame seemingly intractable barriers of power, weight, speed, price, and functionality. Tightly coupled to these advances have been the evolution of human-computer communication: from standing in line with a stack of punch cards, to peering through the virtual horizontal slit of a printer-terminal, to mousing icons on a video monitor onto simple handwriting.

Is there a limit to all this? Yes. And for conventional silicon processes, this limit is already in sight. However, my sense is that conventional digital logic is good for at least 8 rounds of the Gordon Moore diet plan (down to 10 nm feature sizes). And we seem certain to acquire additional sensing capability along the way.

The Paintable Mode

All this stands us before the question: "*When we pass through the shrink that leaves us with full featured, environmentally aware, autonomous processors, each squeezed into a pinhead and sold in bulk, what new natural modes will appear and what will they look like*?" There are certain to be multiple new forms. But there are at least three reasons why I believe that the *paintable* will emerge as a mode distinct from the others.

1) *The incremental cost per MIP will plummet*. As the perennial scarce resource, compute capacity has always been the dominant cost item in the design of personal computing systems. As compute capacity becomes cheaper and cheaper relative to the other design criteria, architectural efficiency will be deemphasized in favor of other concerns such as human affordances. And it is this increased accent on human affordances which favors the ubiquity of a *paintable*.

2) *The marginal cost of adhering to existing programming models will become intolerably high*. The claim of vanishing unit cost for MIPs is moderated by the caveat that we do not yet have good techniques for programming large ensembles of asynchronous processors. Since the days of the first microprocessors, we have always paid a price for this shortcoming; a price which has risen steadily. While we may never obviate the need for the Touring model, the search for a new programming model tailored to a dense, decentralized architecture will become increasingly cost driven.

3) *Given any amount of MIP's, some software applications developer will find a way to over tax them*. This claim is speculative and somewhat at odds with the claim #2 above. Namely that designers will trade MIP's against HCI related affor-

IC (d)Lemma:

For any real N, where N is the maximum available compute capacity on a consumer PC, some software developer will introduce a popular application optimized for use on machine with capacity N+ε.

dances. Still, there is no evidence that the society's appetite for compute cycles is waning. And as the MIP's become available, software developers will seek interesting, useful ways to employ them. Invariably, some applications will be targeted for use on the most powerful platforms commonly available.

This final point has particular import for the wearables community, where the availability of power constitutes a hard upper bound. For any given amount of processing which can be carried or worn, the stationary local environment will always support more. Bluntly, if someone squeezes a PowerPC 6-0-zillion into a nose ring, 50 of them can always be networked and embedded into a lamp shade. A similar dynamic is at work with the price: where the densest, most power frugal MIP's are the most expensive. So, while you will always be able to use what you can carry, you will unlikely be able to carry all you can use.