01 Sep 2008

Hardware Implementations of Swarming Intelligence -- A Survey

Cameron Eric Johnson

Parviz Palangpour

Ganesh K. Venayagamoorthy
Missouri University of Science and Technology

2008 IEEE Swarm Intelligence Symposium
St. Louis MO USA, September 21-23, 2008

Hardware Implementations of Swarming Intelligence – A Survey

Cameron Johnson, *Student Member, IEEE*, Ganesh K. Venayagamoorthy, *Senior Member, IEEE*, and Parviz Palangpour, *Student Member, IEEE*

***Abstract*—Swarming Intelligence (SI) is the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent global patterns to emerge. Because there are no central processing requirements, SI is ideal for parallelization, which lends it well to hardware implementations in many inexpensive processors. Several implementations in hardware that exploit this property for rapid calculation and inexpensive construction are highlighted in this paper to provide a good starting point for developing hardware platforms for SI algorithms.**

I. Introduction

SWARMING Intelligence (SI) is an emergent behavior of multiple unsophisticated agents interacting with each other and their environment. Among the most common forms of swarming intelligence are Particle Swarm Optimization (PSO) [1], [2], Ant Colony Optimization (ACO) [3], [4], Honey Bee swarming [5], and bacterial foraging [6]. Hardware implementations on parallel systems can greatly speed up SI algorithms by dividing the independent agents up amongst multiple processing units. To the knowledge of the authors, there have not yet been any hardware implementations of honey bee swarming or bacterial foraging algorithms.

In this paper, several recent hardware applications and implementations of SI are surveyed and analyzed, including hardware optimization, hardware implementations of PSO and ACO algorithms, and real-world implementations of SI for sensor systems and other mobile robotics applications. By compiling several of the most recent hardware applications and implementations of SI here, it is hoped that this paper may serve as a starting point for any research into advancing SI in the hardware realm, especially in the area of real-time applications. Due to space limitations, this paper focuses mainly on PSO and ACO, leaving discussion of honey bee swarming and bacterial foraging algorithms for another work. The paper is presented in the following format: Section II details the two primary swarming intelligence algorithms, PSO and ACO. Section III examines hardware implementations of PSO designed for use in the training of neural networks. Section IV contains information on the use of PSO implemented for controller design and tuning. Section V deals with mobile robot systems and wireless sensor networks, and section VI covers several other miscellaneous hardware platforms that are promising for SI implementation. Section VII is the conclusion, and finally, Section VIII contains references.

Manuscript received June 15, 2008.

The funding from the National Science Foundation (USA) CAREER Grant ECCS #0348221 and the US Dept. of Education GAANN program, award #P200A070504-08, is gratefully acknowledged for this research by the authors.

C. Johnson, G. K. Venayagamoorthy and P. Palangpour are with the Real-Time Power and Intelligent Systems Laboratory, Missouri University of Science & Technology, Rolla, MO 65409 USA (email: cej@mst.edu, gkumar@ieee.org and pmpv3b@mst.edu).

II. PSO and ACO Algorith

PSO and ACO are perhaps the two most well-known SI algorithms in use today. They both are inspired by flocking/swarming behavior of insects – most notably ants in the case of ACO – fish, and birds, and they are highly versatile CI paradigms.

A. PSO

Among the most common forms of SI is PSO [1], [2], whose general algorithm is illustrated in fig. 1. It is a swarm of individual particles designed to search a "solution space" by "flying" through it, testing each particle's fitness according to an optimization function every iteration before adjusting its trajectory and moving again. The position within the search space is a set of coordinates which record the solution that the particle represents. It has velocity and momentum at each iteration updated according to (1) & (2), and two loci (personal best and global best) which exert force to pull it into a new trajectory. The values rand1 and rand2 are random numbers between 0 and 1. The constant w is an inertia value, typically about 0.8, while c_1 and c_2 are

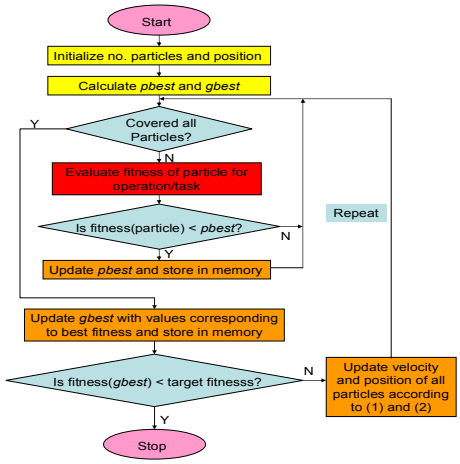

Fig. 1. PSO algorithm flowchart

978-1-4244-2705-5/08/$25.00 ©2008 IEEE

typically about 2, but can be varied to find optimal performance. Each particle stores its personal best position, and the global best position found by any particle is stored globally.

$$\vec{x}_i(k) = \vec{x}_i(k-1) + \vec{v}_i(k) \quad (1)$$

$$\vec{v}_i(k) = w \cdot \vec{v}_i(k-1) + c_1 \cdot rand_1 \cdot \left(\overrightarrow{pbest_i}(k-1) - \vec{x}_i(k-1)\right) + c_2 \cdot rand_2 \cdot \left(\overrightarrow{gbest_i} - \vec{x}_i(k-1)\right) \quad (2)$$

B. ACO

ACO is a form of swarming intelligence based on the social behavior and routing techniques of ants [3], [4], whose general algorithm is illustrated in fig. 2. It uses, in its most abstract sense, individual agents to traverse a graph from node to node in an effort to find the shortest path that completes the route. Each agent chooses, at each node, an edge of the graph along which to travel. The choice is random, with probability distributions determined by factors set up by the user. Often, it is initially a uniform distribution.

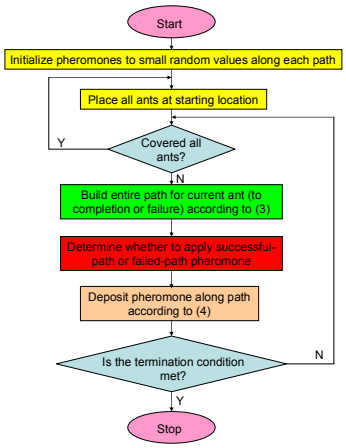

Fig. 2. ACO algorithm flowchart.

Once an agent – analogous to an individual ant – reaches the destination node or a dead end, it makes a return trip to the starting point, dropping digital "pheromone" along its path to mark it. This pheromone alters the probability Φ, according to (3), of another "ant" choosing that particular path at a given node wherein a choice is presented. $\Phi_{ij,k}$ is the probability of a path to node i being chosen at node j by ant k, and is determined by the amount of pheromone on the path to node i from node j relative to the total amount of pheromone on all paths leading away from j toward nodes which ant k has not yet visited. $C_{i,k}$ is the set of nodes adjacent to node j which ant k has not yet visited, and α is a constant less than 0 and β is a constant less than 1. They are used to tune the algorithm [7].

If the destination was reached, the pheromone dropped increases the likelihood that other ants will choose that path. If the path led to a dead end, the pheromone dropped decreases the likelihood an ant will choose that path. The "desirability" η_{ij} of a path is often the inverse of the distance of the path, and used to simulate, in algorithms (like the one shown in Fig. 2) where each ant's entire route is handled at once before moving on to a second ant, the delayed pheromone-dropping effect of ants working simultaneously in parallel. The amount of pheromone τ_{ij} on each path between nodes is controlled at each stage by (4), with a decay in pheromone levels controlled by the constant ρ, and an addition of pheromone $\Delta\tau_{ij}(t)$ equal to the sum of all pheromone dropped on that path in a given iteration.

$$\Phi_{ij,k}(t) = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{c \in C_{i,k}} [\tau_{ic}(t)]^{\alpha}[\eta_{ic}]^{\beta}} \text{ if } j \in C_{i,k}$$

$$\Phi_{ij,k}(t) = 0 \qquad \text{if } j \notin C_{i,k} \quad (3)$$

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \quad (4)$$

Because pheromone is dropped either in greater amounts or earlier (or both) along shorter paths, dead-ends are more quickly ruled out, and shorter successful paths have an increased chance that a later ant will take that route. Likewise, however, pheromones slowly "evaporate" (according to the constant ρ, which is a value between 0 and 1 [7]) with time, which means a path neglected for too long will lose any markings it might have. So the more desirable a route is, the more pheromone is laid on it and the greater chance more ants will take it. This leads to reinforcement of valid, desirable paths and neglect of invalid or less desirable paths, but leaves room for experimentation thus potential for improvement.

III. PSO FOR NEURAL NETWORK APPLICATIONS

Neural Networks are a computational intelligence (CI) paradigm all their own, but PSO has been successfully used in conjunction with them for several purposes.

PSO has been found to be a highly effective means of training neural networks (NNs). Each individual particle is a candidate NN, and the coordinates within the solution space are the synaptic weights between the neural layers. The authors of [8] and [9] used such a method to train a feed-forward multi-layer perceptron (such as the one shown in Fig. 3) on a Xilinx Virtex2P FPGA [8]. They designed a hardware PSO core detailed in [9] which handled all particle functions and a temporary storage system consisting of two memory blocks for temporary storage of weights. While the PSO updates and stores a new set of weights in one memory block, the NN runs to generate its fitness for use in updating the next iteration of the PSO.

The neuron architecture performed competitively with more standard software implementations, and while it was a bit slower in some instances, it had higher precision. The system's performance could be easily improved by adding more neuron processing elements (by taking up more slices

in the FPGA) or by upgrading to a faster FPGA. Of particular interest in this design is that, once trained, the PSO elements can be reconfigured to serve as part of the trained neural network's processing elements, leaving a trained network in place without requiring additional hardware.

SI is also quite useful for retraining a neural network in an adaptive environment. For instance, controlling multiple data signals in a power line for use in power line communication requires that each of the signal frequencies on the line be distinctly detectable. In [10], a design for a hardware NN is implemented that can distinguish between multiple peaks in frequencies which represent the individual channels. Unfortunately, without careful control of the load impedance on the sensors, the signals tend to cluster unpredictably, which causes interference between the data signals. An adaptive PSO is used in real-time in [10] to test and control the load impedance signal spikes at the sensors are more evenly distributed, thus making the NNs much more robust in interpreting multiple signals and sorting the data.

Perhaps more interestingly, PSO can also be used to invert a neural network [11]. One situation where this is useful is simulation of sonar in a given environment. Computationally intensive models can determine what dB level of sonar will be in every location of the environment (broken down into grid squares) with high accuracy given the placement of sonar sensors. A neural network can greatly enhance the speed of this simulation, using the sonar pulse loci as the input layer and the map grid locations as the output layer (outputting the dB measures at each grid square). A properly trained NN can institute such a simulation in microseconds. However, the information desired is where to place the sensors to obtain a desired dB level at each grid location. In other words, rather than training the network such that a known set of inputs (x_i variables in Fig. 3) generates a desired set of outputs (y_i variables in Fig. 3), the desired information is what inputs to use on a given set of weights (w_{ij} variables in Fig. 3) to generate a desired output.

A PSO is implemented on the Xilinx XC2V6000 in [11], using the NN (implemented on another FPGA of the same make and model) as the fitness function. In software testing, the PSO was found to be less than 2 dB off of the desired patterns, but took nearly two minutes on a 1.2 GHz processor to find these solutions. The FPGA implementation (which ran at 100 MHz) of the NN runs about 60 times faster than the software simulation. Shortening the time it takes to perform fitness evaluations 60-fold is already a dramatic increase in performance rate.

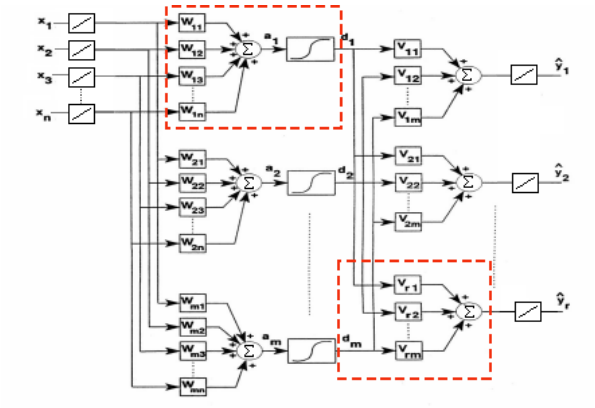

Fig. 3. MLP Network with a hidden layer neuron and an output layer neuron highlighted.

In order to properly implement the PSO on the FPGA, certain alterations were necessary, and are detailed in [11]. Of particular note, random number generation proved to be an issue. Three approaches to random number generation were tried in [11]: a linear left shift register; a squaring method where the "random" number is the fractional portion of the square; and simply not generating a random number at all (i.e. a deterministic swarm).

In software simulation, the deterministic swarm performed very poorly. However, in hardware, it actually out-performed the software stochastic swarm's accuracy by about a tenth of a dB. In all, the shift from software to hardware resulted in insignificant increase in error between desired output and best found solution. The hardware implementation takes 1.8 seconds, which is a 60-fold reduction in the time from the software implementation. While this is still not quite fast enough for real-time, the latest Xilinx FPGAs that operate at 500 MHz can reduce the time to a mere 0.36 seconds with an identical implementation.

IV. PSO for Controller Design and Tuning

Controllers are complicated devices which need to monitor several factors and react to them quickly by adjusting parameters to other devices. SI can be used to tune them initially, and, perhaps more interestingly, to retune them as the environment changes.

PSO has been implemented on DSP boards for the purpose of minimizing oscillations in a power system in [12] and [13], designing and tuning the controller parameters in an adaptive and reactive fashion that leaves them robust against rapid changes and oscillations in the systems they control.

An electrically powered naval vessel with multiple sources of pulsed power requirements can keep its power levels relatively stable by implementing generator field excitation control coupled with immediate energy storage devices to meet the pulsed energy needs and then slowly recharge [12]. An online design for an optimized excitation controller is proposed in [12], utilizing a PSO implemented

on a DSP to minimize the voltage deviations when high power pulsed loads are directly powered from DC elements in the ship's power system.

To simulate the hypothetical ship's power systems in the lab, a smaller-scale model was set up using a small-scale three-phase 60 Hz 5kV asynchronous generator and a 15kW DC motor to apply mechanical torque to a synchronous generator. The laboratory model of the excitation controller consists of a sensor board, an A/D converter, an MSK2812 DSP board, and a D/A converter. The laboratory setup is shown in Fig. 4.

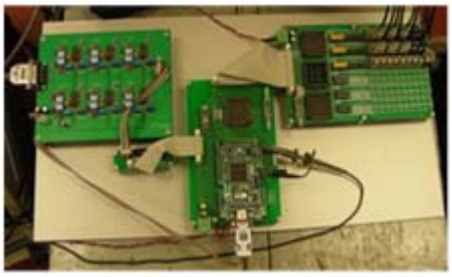

Fig. 4. Experimental Setup for Scaled Model of Hypothetical Ship's Power System [12].

The four parameters being tuned by the PSO are three time constants (T_a, T_b, T_c) and a gain constant (K_a), which must be carefully selected to give the excitation controller (Fig. 5) satisfactory performance under normal and pulsed load conditions. The sampling rate used in the laboratory was 1s, starting when the pulsed load is removed, and the sampling period was 2ms, leading to a total of 500 sampled points. Twenty particles were used in the experiment, and their fitnesses were determined by the settling time of the system oscillations after a disturbance, with better fitness going to shorter settling times. The PSO-controlled oscillations damped out in hundreds of milliseconds or less (Fig. 6), while a normal control system bounced five or six times over nearly 1000 ms before settling.

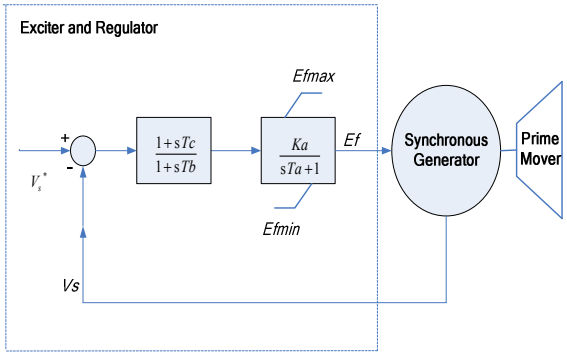

Fig. 5. A block diagram for synchronous machine excitation control system [12].

Another DSP implementation of PSOs experimented with Power System Stabilizers (PSSs) simulated on a Real Time Digital Power System Simulator (RTDS), shown in Fig. 7 [13]. This experiment utilized a TMS32067 DSP integrated in the Innovative Integration M67 to implement the two PSSs used. Each PSS has six time constants and a gain constant, for a total of seven parameters to be tuned per PSS, or fourteen parameters in each particle. The objective fitness function of each particle is determined by reading the transient energy out of two generators, G1 and G3, in post-fault operating conditions. The fitness function ultimately also incorporates the settling time of the transient energy, as well as its overall magnitude. Two testing loads were used, 967 MW and 1167 MW, for one operating mode, and 1100 MW and 1600 MW for a second operating mode. In both operating modes, G1 stabilized fastest, but both G1 and G3 stabilized much faster and exhibited less transient energy than the untuned system. So [13] successfully implemented real-time PSO tuning of a PSS in DSP hardware.

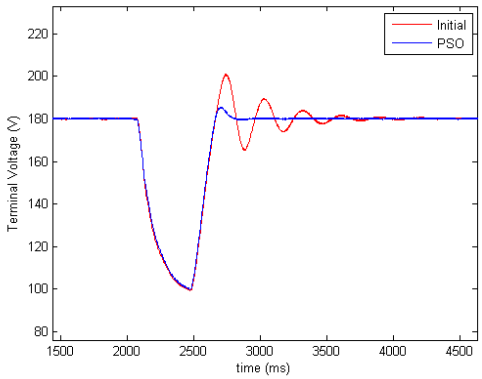

Fig. 6. Pulse recovery with 4.86kW pulse and 0.4s duration with terminal voltage [12].

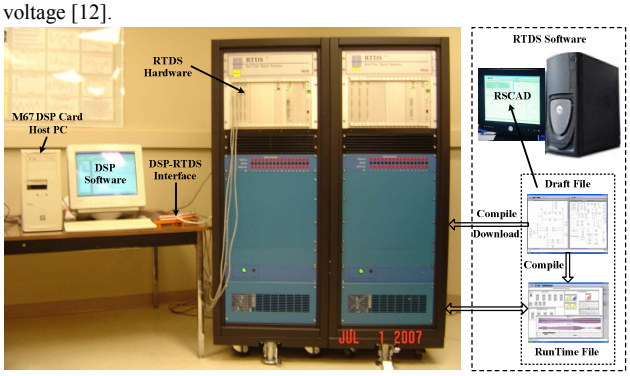

Fig. 7. Laboratory setup with RTDS and DSP for real-time power system studies with computational intelligence paradigms [13].

V. SI for Mobile Robots and Sensor Networks

Mobile Robot swarms and networks of individual sensors are an obvious place to implement SI: they consist of physical independent units, and each can be directly analogous to a swarming agent.

A. ACO in Wireless Sensor Networks

Perhaps one of the most obvious implementations of an SI algorithm in hardware is one whose physical structure closely parallels the conceptual architecture of the algorithm. One such scenario is data routing in Wireless Sensor Networks (WSNs). Because wireless sensor nodes are largely unattended, low-power and low-cost devices, power consumption is of significant concern to their useful lifespan. One method of minimizing power consumption is to minimize individual transmission distances, transferring collected data from node to node back to the base station.

This architecture, combined with the problem of finding the lowest-cost path back to the base station, is ideal for an ACO implementation.

Such an algorithm is implemented in [14] using stationary wireless sensors as the nodes and the paths between them as potential directions to travel. "Ants" are implemented as data packets sent to nearby nodes with a probability to choose any given path based on several factors. Initial probability distributions are based on power consumption requirements from the current node to the next node. The base station, not having any particular power constraints, can broadcast updates globally based on the information carried in by the data package ants. Because the base station can't tell if an ant was sent but never made it, there is no "bad path" pheromone, but an evaporation coefficient slowly degrades the probability of paths being chosen over time, so any path that did not successfully lead to the base station will slowly lose favorable chances of being taken. In order to prevent a data packet from getting lost in an interminable loop between wireless sensor nodes, each ant also retains a memory of what nodes it has visited before. If a node receives an ant which has visited it already, it will not write that ant into its memory for retransmission, and the ant will have to go to a different node. This also conveniently prevents back-tracking.

In this application, hardware implementation happens naturally, as the nodes and paths are literal things in the real world, with the nodes themselves serving as parallel hardware units. The analogy to ACO is not perfect, as there is some redundancy as ants may be received and retransmitted by more than one node (resulting in some minor duplication), but this redundancy was found to be minimal in impact on power consumption and ensured a more rapid and reliable transmission of data back to the base station, with less sensitivity to the loss or failure of sensor nodes due to the ability to reroute in real time.

Another SI-based WSN implementation, designed for military scouting and target tracking applications, is described in [15] and uses a variant on ACO wherein digital pheromones are laid on areas of interest found by individual agents, and reinforced if other agents find them and agree, or allowed to evaporate if agents determine them to be of less interest than indicated (by not reinforcing them upon return visits). Different "flavors" of pheromones can be implemented to represent different kinds of information, or even to add new functionality to the algorithm. This implementation proved to be capable of a wide variety of applications, including area surveillance, area-of-interest observation, and target acquisition and tracking. It exhibited complex self-organization to divide labor evenly between agents in surveying the target area and focusing on the interesting pieces.

B. PSO in Collective Robotic Search

A similarly direct application for PSO in hardware is presented in [16], where robotic platforms are used as literal swarming agents searching a physical search space. Created for use in hazardous environments or for dangerous materials, the design goals called for the robots to be individually "small, inexpensive, and dispensable" [16].

The pre-existing iRobot Roomba was chosen as the base platform for the individual robots in the swarm, and the experiment was done with a population of five such robots, which already had built-in sensor systems and motion controls. The details of the wireless communication are fully explained in [16], and the overall hardware system is shown in Fig. 8. The system is broken into two parts: a control module whose two primary duties are to initialize the network so the robots can join, ensuring all communicate properly, and to correlate the robots' positional data and fitness to calculate the global best fitness (gbest) and broadcast it to all members of the swarm, and the robots themselves, who handle their own individual positional updates and calculations and keep track of their own personal best fitnesses (pbest), sending updates of their pbests to the control module when they change.

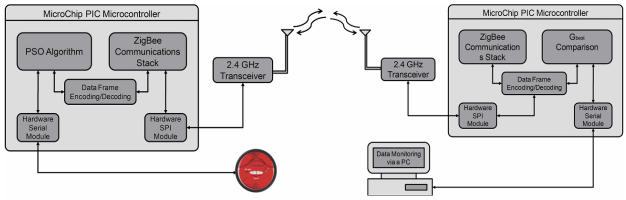

Fig. 8. Schematic block diagram for the hardware implementation of the collective search robotic platform [16].

Experiments run on this system were reported in [16] to demonstrate accurate and swift convergence on real-world physical targets in the search space, demonstrating that a direct translation of the PSO algorithm to real-world physical search tasks is viable and useful. Ultimately, this should not be surprising, since the PSO algorithm is modeled on real-world birds flocking and fish schooling [1].

One requirement of such systems, however, is an initial distribution. The experiment in [16] did this manually, but [17] suggests three methods for automating their distribution from a single location, all of which require minimal communication.

Another of the earlier implementations of mobile robot swarms was a design for adding mobile sensors to a static sensor network [18]. Unmanned aerial units (UAVs) with multiple wireless radios serve as the mobile sensors. One radio is used to talk to other UAVs, another is used to talk to static sensor nodes and/or a base station, and a third can be installed to talk to a satellite for relaying data.

Lacking a satellite, a backbone of swarms can be used for data forwarding back to a base. Backbone design relies on one member of a swarm being named "Swarm Leader", and receives a hardware radio with a longer range than normal. It is used to contact swarm leaders of nearby swarms, using a multi-hop ad hoc network to relay information back to base.

As sensor swarms move more into fully autonomous robotic units, the power consumption becomes of greater concern, both due to miniaturization causing a shrinking of

available space for batteries, and due to lack of stationary sensors to back up failing mobile units. One solution is presented in [19], dividing the sensor units into "perimeter" and "core" nodes. The framework, called "SenseSwarm", has three conceptual phases: Perimeter Construction, Acquisition, and Replication.

Perimeter construction involves determining the relative locations of the mobile sensors, and divides them into "perimeter" and "core" nodes based on this positioning. The philosophy behind this sensor design is that interesting changes to the environment will only happen on the perimeter, and anything internal to that will remain much the same as when it was last detected. This phase is run every few cycles to ensure that new perimeter nodes are appropriately assigned as the swarm changes its physical configuration. Perimeter nodes perform all sensing duties, while core nodes save energy by performing data storage and processing duties rather than operating their external sensors.

The acquisition phase is constantly running on all perimeter nodes, as they collect data. Replication is a phase run every few cycles just before the next perimeter construction phase, and involves perimeter nodes replicating their data into nearby neighbors, helping to determine the next perimeter/core division.

Overall, this sensor network proved in testing to be equal to more uniform structures in data acquisition, but demonstrated up to a 75% energy savings over the uniform variant.

C. PSO for Fault-Tolerant Sensor Systems

PSO can also be used in conjunction with an autoencoder to create fault-tolerance in a sensor encoding scheme. In [20], an autoencoder is described as a Multi-Layer Perceptron (MLP) NN with as many inputs and outputs as there are sensors in the system, and fewer neurons in the hidden layer than there are sensors in the system. Relying on the fact that some of the data from the sensors is redundant, the autoencoder uses the reduced dimensionality of the hidden layer to correlate the redundant data and return as output the same information it receives on its input. The autoencoder is shown in Fig. 9, where X are the incoming signals to the sensors, S is the input to the system as well as the comparison of desired results, and $\hat{S}$ is the output of the sensor system.

This redundancy can then be exploited to minimize the impact of failed sensors in the system by the inclusion of a PSO on the inputs of the autoencoder. S_H and S_R are the healthy inputs from the sensors and the outputs from the autoencoder, respectively. S_E is the error between $\hat{S}_H$ and S_R and is zero (or at least below the threshold of error tolerance set up when the system was trained) as long as all sensors are working.

When one (or more) of the sensors stops working, however, S_H and S_R will no longer be equal, and S_E will become significantly non-zero. The remaining healthy inputs are then fed into the auto-encoder along with an estimate of the missing sensor data S_M generated by the PSO. The PSO uses S_E as a fitness measure to adjust S_M until S_E is once again negligible.

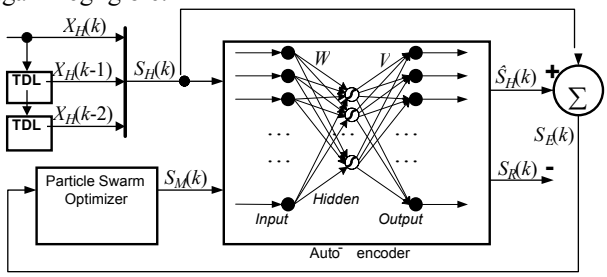

Fig. 9. Overall structure of the MSR with on-line restoration of missing sensor data. TDL denotes "time-delay lock" [20].

As long as the number of functioning sensors exceeds the dimensionality of the hidden layer of the autoencoding MLP, the redundancy will keep the system functioning with reasonable levels of accuracy. This system was implemented and tested in hardware on the RTDS (with hardware specifications as given earlier) shown in Fig. 7. A disturbance in the power supply or draw causes a sudden change in necessary operating mode, and the sensors are used to control the fluctuations and adapt to the new situation.

Fig. 10 illustrates the performance of the power regulator as controlled by this system with zero, two, and three current sensors missing. The close similarity of the performance with all three conditions indicates how well this missing-sensor replacement works in real-time.

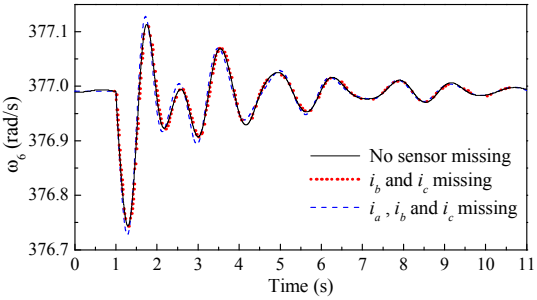

Fig. 10. Dynamic performance of a power regulation system with different (missing) sensor tests during an increase of electric load [20].

VI. Other SI Hardware Platforms

There are far too many parallel hardware platforms to cover in the limited space of this paper, but a few are mentioned here, including Grid Computing, ACO on FPGAs, and multi-agent systems.

A. Grid Computing

Grid computing is another parallel system demonstrated to be useful for implementation of SI. Grid computing applies the metaphor of a power grid to computing, connecting computers over a wide area together, and letting users plug into the entire system without worrying about where, specifically, the computational resources are coming from [21].

A multi-objective PSO (MOPSO) is implemented in [21] via grid computing. Swarms are repeatedly divided into

subswarms, which run for a limited amount of time and return their result to a central server. The next iteration includes a "guide" in the form of a functional particle to re-start the search with all other particles re-initialized within the local search space.

Two methods of selecting guides are proposed: cluster-based (designed to function on a fixed number of similar processors, and hypervolume-based (designed to operate on a heterogeneous set of processors) [21]. Both algorithms out-performed more traditional implementations of the MOPSO on single processors.

B. ACO in Hardware

The majority of work with ACO has been done in software implementations on problems whose structure lends itself to ACO's structure and style of problem solving. [22] utilizes it in perhaps the most obvious test problem: the traveling salesman. However, there are several hardware applications that logically lend themselves to this paradigm's quirks. One such direct hardware implementation is presented in [23], which is an attempt to speed up the algorithm by taking advantage of its parallel capabilities and make it more useable in real-time. The algorithm is broken down into several key segments:

1) Initial Setting of a Parameter
2) Read Data
3) Calculation of Distance
4) Sorting
5) Hardware
6) Finish.

The hardware section is further broken down into sub-steps comprised of one loop nested within another. The outer loop includes an initialization step to determine how many iterations are to run and where each of the active ants will start. The inner loop exploits the parallel nature of ACO to greatly increase the algorithm's speed, with multiple ants running at once, using the individual local update rule to traverse the route from node to node until termination conditions are met. Once the inner loops are done, the outer loop updates pheromone levels, checks for global termination conditions, and repeats if termination conditions are not met.

In experiments, the ACO algorithm (both simulated entirely in software and implemented in hardware) outperformed a GA used as a control, and the hardware implementation of the ACO algorithm achieved speeds up to six times faster than the software simulation.

An FPGA implementation discussed and tested in [24] lists some key restrictions FPGA hardware places on ACO:

1) Pheromone values and random numbers take floating point representation which is too fine-grained for FPGAs.
2) Multiplication operations required by evaporation and heuristic integration operations are not easily supported by most FPGAs.
3) Space and time complexity is too large for modern FPGAs due to need to calculate prefix sums in numerators over as-yet unchosen items in the selection set.

To circumvent these difficulties, a high-level mapping of ACO to an FPGA design is proposed in [24] using a series of modules. The population module contains a matrix of the individuals, with the best fit individual at the top of the list and a first-in-first-out (FIFO) matrix containing the rest. The Generator module sends individuals out one after another to explore the solution space, where the Evaluation module waits to send those who have completed their path for this iteration back to the Population module with pheromone updates for the path they took, as well as the fitness of the path they took. If one comes back with a higher fitness than the current best, the new one is placed at the top of the list. The route-traversal takes advantage of the parallel nature of the FPGA by allowing several ants to be exploring at once, but solving the floating-point precision difficulties still remains an issue.

C. Multi-Agent Systems and SI

Multi-agent systems are related to SI in that both utilize multiple relatively simple and independent agents to achieve complex emergent behaviors. The only difference is that multi-agent systems allow differences between their agents, whereas SI agents are all homogeneous. It would be remiss to conclude this paper without giving some examination to the work done in implementing hardware multi-agent systems.

The majority of multi-agent work, however, has been done in the development of software agent design to create intelligent emergent behavior in a single computer system [25], [26]. In particular, [25] examines in detail protocols and design rules for software agents useful in controlling power systems, while [26] explains a programming language called "*New Valid*" for programming multi-agent systems, including the creation of new agents dynamically. Promising work on multi-agent control mechanisms for hardware-based mobile robots is shown in [27], where successful MatLab simulations of their programming language designed to control multiple mobile robot agents have been performed, but as of yet the specific problems peculiar to hardware processing have yet to be addressed by the authors of said language.

However, successful completion of the complex task of jointly carrying a load between three independent robot agents through a narrow opening was accomplished in [28]. Physically identical, one robot was designated "leader", which amounts to being responsible for primary path discernment, while the other two were "followers". All three robots were equipped with tactile sensors to detect the balance and lateral force of the object they jointly carry on their backs and IR sensors for environmental proximity detection and, in the leader's case, beacon-finding. They

worked to maintain a stable hold with minimal lateral force, which kept them in the proper orientation without any sophisticated communication systems. Once the leader detected the beacon, it began moving in that direction. The other robots' maintained proximity to the leader due to the need to keep the object stable, and the followers used their proximity detection to avoid running the whole group into the wall, correctly orienting themselves so that the whole team and their attendant object passed safely through the opening.

While [28]'s implementation is not an SI algorithm, it provides insight into the kinds of simple behavioral controls that can be performed on physical agents, which is applicable in the SI field.

VII. Conclusion

Swarm Intelligence algorithms are robust systems easily executed in parallel, making them ideal for hardware implementation in inexpensive but plentiful processors, such as FPGAs, GPUs, or Grid Computing systems. The very nature of their metaphor lends them well to mobile robot swarms and wireless sensor network control. Several recent promising implementations have been highlighted in this paper, demonstrating the power and versatility of this paradigm. Hardware implementations greatly enhance the speed and performance of these algorithms, and it is hoped that implementations such as those shown here can be used to make real-time Swarm Intelligence a feasible advancement in the near future.

References

[1] J. Kennedy, R. Eberhart, "Particle swarm optimization," Proceedings of IEEE International Conference on Neural Networks, Vol.4, Perth, Australia, 1995, pp. 1942-1948.

[2] Y. del Valle, G. K. Venayagamoorthy, S. Mohaghenghi, J.C. Hernandez, R. G. Harley, "Particle swarm optimization: basic concepts variants and applications in power systems," IEEE Transactions on Evolutionary Computation, Vol. 12, No. 2, April 2008, pp. 171-195.

[3] A. Colorni, M. Dorigo, V. Maniezzo, "Distributed optimization by ant colonies," ECAL91, European Conference on Artificial Life, Paris, France, 1991, pp. 134-142.

[4] M. Dorigo, V. Maniezzo, A. Colorni, "The ant system: optimization by a colony of cooperating agents," IEEE Transactions on Systems, Man, and Cybernetics, 1996, pp. 29-42.

[5] K. Weber, S. Venkatesh, M. V. Srinivasan, "Insect inspired behaviours for the autonomous control of mobile robots," Proceedings of the 13th International Conference on Pattern Recognition, Vol. 1, Aug. 25-29 1996, pp. 156-160.

[6] K. M. Passino, "Distributed optimization and control using only a germ of intelligence," Proceedings of the 2000 IEEE International Symposium on Intelligent Control, July 17-19, pp. 5-13.

[7] V. Maniezzo, L.M. Gambardella, F. De Luigi, "Ant colony optimization, new optimization techniques in engineering," by G. C. Onwubolu and B.V. Babu, Springer-Verlag Berlin Heidelberg, 2004, pp. 101-117.

[8] A. Farmahini-Farahani, S. M. Fakhraie, S. Safari, "Scalable architecture for on-chip neural network training using swarm intelligence," EDAA 2008, Design, Automation, and Test in Europe, March 2008, pp. 1340-1345.

[9] A. Farmahini-Farahani, S. M. Fakhraie, S. Safari, "Sopc-based architecture for discrete particle swarm optimization," Proc. of IEEE Intel. Conf. on Electronics, Circuits and Systems, Marrakech, Morocco, Dec. 2007, pp. 1003-1006.

[10] P. Chanyagorn, H. H. Szu, H. Wang, "Collective behavior implementation in powerline surveillance sensor network," Proceedings of International Join conference on Neural Networks, Vol. 3, Montreal, Canada, July 31 – August 4, 2005, pp. 1735-1739.

[11] P. D. Reynolds, R. W. Duren, M. L. Trumbo, R. J. Marks II, "FPGA implementation of particle swarm optimization for inversion of large neural networks," IEEE 2005, Swarm Intelligence Symposium 2005, pp. 389-392.

[12] C. Yan, G. K. Venayagamoorthy, K. Corzine, "Hardware implementation of a PSO based online design of an optimal excitation controller," SIS 2008, Swarm Intelligence Symposium, St. Louis, MO, Sept. 2008.

[13] P. Palangpour, P. Mitra, S. Ray, G. K. Venayagamoorthy, "DSP PSO implementation for online optimal controller design," AHS 2008, NASA/ESA Conference on Adaptive Hardware and Systems, Noordwijk, the Netherlands, 2008.

[14] S. Okdem, D. Karaboga, "Routing in wireless sensor networks using ant colony optimization," AHS'06, Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems, 2006, 401-404.

[15] J. A. Sauter, R. Matthews, H. V. D. Parunak, S. A. Brueckner, "Performance of digital pheromones for swarming vehicle control," AAMAS '05, Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems, July 25-29 2005, Utrecht, Netherlands, 2005 ACM, pp. 903-910.

[16] P. Palangpour, C. Parrott, L. Grant, "Collective robotic search in software and hardware implementation," IEEE Industry Applications Magazine, MAR/APR 2008, pp. 71-73.

[17] M. Siebold, J. Hereford, "Easily scalable algorithms for dispersing autonomous robots," IEEE Southeastcon, April 3-6 2008, pp. 545-550.

[18] M. Gerla, K. Xu, "Multimedia streaming in large-scale sensor networks with mobile swarms," SIGMOD Record, Vol. 32, No. 4, December 2003, pp. 72-76.

[19] D. Zeinalipour-Yazti, P. Andreou, P. K. Chrysanthis, G. Samaras, "SenseSwarm: a perimeter-based data acquisition framework for mobile sensor networks," Proceedings of the 4th International Workshop on Data Management for Sensor Networks (DMSN'07), Vienna, Austria, 2007, pp. 13-18.

[20] W. Qiao, G.K. Venayagamoorthy, "Missing-sensor-fault-tolerant control for SSSC FACTS device with real-time implementation," IEEE Transactions on Power Delivery, in print.

[21] S. Mstaghim, J. Branke, H. Schmeck, "Multi-objective particle swarm optimization on computer grids," GECCO'07, July 7-11, 2007 ACM.

[22] M. Dorigo, L.M. Gambardella; "Ant colony system: a cooperative learning approach to the travelling salesman problem," IEEE Transactions on Evolutionary Computation, Vol. 1, No. 1, 1997, pp. 53-66.

[23] M. Yoshikawa, K. Terai, "Architecture for high-speed ant colony optimization," IEEE 2007, International Conference on Information Reuse and Integration, 2007, pp. 1-5.

[24] H. Duan, X. Yu, "Progresses and challenges of ant colony optimization-based evolvable hardware," WEAH 2007, Proceedings of the 2007 IEEE Workshop on Evolvable and Adaptive Hardware, 2007, pp. 67-71.

[25] S. D. J. McArthur, E. M. Davidson, V. M. Catterson, A. L. Dimeas, N. D. Hatziargyriou, F. Ponci, T. Funabashi, "Multi-agent systems for power engineering applications—part II: technologies, standards, and tools for building multi-agent systems," IEEE Transactions on Power Systems, Vol. 22, No. 4, November 2007, pp. 1753-1759.

[26] M. Amamiya, S. Kusakabe, S. Aramaki, "A multi-agent system description language *new valid* and its application to robot control," IEEE 1993, IEEE 2nd International Workshop on Emerging Techniques and Factory Automation, Sept. 1993, pp. 27-29.

[27] S. Ahmed, M. N. Karsiti, "A testbed for control schemes using multi agent nonholonomic robots," IEEE EIT 2007, IEEE International Conference on Electro/Information Technology, 2007, pp. 459-464.

[28] M. R. Ahmad, S. H.M. Amin, R. Mamat, "Development of decentralized based reactive control strategy for intelligent multi-agent mobile robotics system," ICARCV'02, Seventh Ihnternational Conference on Control, Automation, Robotics, and Vision, Vol. 1, Dec 2002, Singapore, pp. 220-227.