

# Using Randomly Assembled Networks for Computation

Andrey Zykov and Gustavo de Veciana

Department of Electrical & Computer Engineering  
The University of Texas at Austin

**Abstract.** This paper makes the case for perturbation-based computational models as a promising choice for implementing next generation ubiquitous information applications on emerging nanotechnologies. Our argument centers on showing such computational engines could be suitably designed for, and implemented on, technologies with low manufacturing precision and high defect densities and performance uncertainty which are likely for such technologies. A perturbation-based approach to computation on real-time streams dictates a new platform and design principles. We propose a representative hybrid computing platform which combines a *core* dynamic network, based on nanodevices, combined with a more reliable CMOS *readout* layer. Cores need only exhibit sufficiently rich dynamics to capture the input stream's characteristics, but do not require detailed synthesis making them robust to defects, i.e., could be randomly assembled networks. Programming in this context is replaced with a configuration (learning) step which leverages the reliable CMOS layer and core's dynamics to approximate the desired function. This paper discusses some of the possible advantages and pitfalls of this approach, along with interesting design principles, e.g., unstructured redundancy, core sharing, spatial decomposition via weak interconnection, and exploitation of near decomposability of application dynamics.

**Key words:** perturbation, computation model, non-Turing, embedded systems, signal processing, cognitive applications

## 1 Introduction

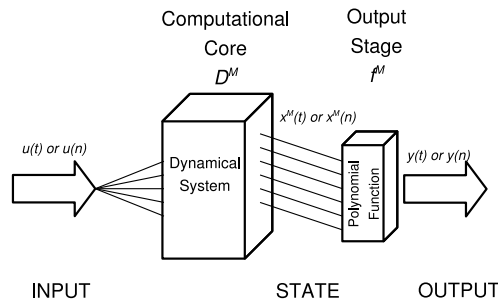
Advances in the synthesis and self-assembly of nanoelectronic devices suggest that the ability to manufacture dense nanofabrics is on the near horizon [1, 2, 3, 4, 5, 6, 7]. Yet, effective ways of utilizing emerging nanoelectronic technologies still elude us. The tremendous increase in device density afforded by nanotechnologies is expected to be accompanied by substantial increases in defect densities, performance variability, and susceptibility to single event upsets caused by cosmic radiation (energetic neutrons) and alpha particles [2, 8, 5]. System-level design adhering to current computational models may thus soon reach fundamental scaling limits, where the increased densities are countered by overheads associated with achieving defect- and fault-tolerant designs that are

robust to performance variability [8, 9, 10, 11, 12]. Thus, it is critical to consider and explore alternative computational models that can operate under such difficult conditions.

Additionally, the nature of next generation ubiquitous information technology (IT) – including many challenging real-time streaming media applications, such as voice and image recognition, as well as a myriad of automation/control and robotics applications – also calls for rethinking current computational models and associated design paradigms. Specifically, in order to enable the massive embedded systems’ deployment required by next generation ubiquitous IT, it is imperative to rely on low design cost/complexity platforms that can be easily configured to implement the many tasks at hand, with acceptable performance. Unfortunately, the cost and complexity of system-level design adhering to current computational models continues to increase dramatically, conflicting with these requirements.

**Contributions.** In this paper, we investigate a promising new class of computational models, called perturbation-based, and show its potential to synergistically address the two sides of the complex system design equation: technology and applications. Our argument on the suitability of this computational model for next generation IT systems targeted at nanotechnologies is based on five main points – the first three relate to technology issues while the remaining address system-level design and application issues. Specifically, as will be seen, the suitability of perturbation-based computing for emerging nanoelectronics (‘eNano’) technologies is predicated on: (1) its reliance on a computational core that can, to a large extent, be ‘randomly assembled’, thus relaxing strict manufacturing precision and stability requirements; (2) its inherent tolerance to manufacturing defects or hard faults – these become simply part of the (desirable) randomness in the structure of the computational core; and (3) its natural robustness to structural noise (i.e., internal noise resulting in topological changes) caused by performance variability/fluctuations, which, as will be seen, can be effectively ‘filtered out’ during the task-dependent machine configuration phase. These three points make perturbation-based computing very promising for technologies exhibiting the high defect densities and substantial performance and structural uncertainty projected for emerging nanoelectronics. At the same time, characteristics of perturbation-based computing that make it promising to address the challenges and needs of next generation IT systems, include: (4) its suitability for implementing the many soft real-time stream processing and reactive control tasks that will comprise such systems; and (5) the limited design effort required, in that, as we will show, this computational model can be realized/implemented on configurable platforms, usable for many different tasks.

To establish the promise and potential of perturbation-based computing, this paper proposes an instance of a hybrid eNano-CMOS platform for realizing such machines – as will be seen, the platform relies on a new style of configuration that, we believe, can directly leverage the strengths and circumvent the limitations of technologies characterized by high density but also high structural and performance uncertainty. We further identify and demonstrate a new set of



**Fig. 1.** A Perturbation-based machine.

fundamental *design principles* and *decomposition strategies* which are effective for perturbation-based computing platforms, and propose a multi-core machine architecture which exposes these principles. The importance and impact of this second set of contributions lies in establishing that this new class of computational models will scale and is amenable to systematic design, two key practicality concerns. These points are empirically demonstrated for a representative set of soft real-time processing tasks from a variety of domains.

Note that the class of computational models investigated in this paper was recently independently discovered by two research groups [13, 14]. Yet, their work was driven by research pursuits and objectives quite different from those in this paper. Section 4 gives details on such prior work and establishes the uniqueness and novelty of our paper’s contributions. In particular the focus of this paper lies not on matching biological systems, but understanding the engineering and design concerns.

## 2 Background: The Principles of Perturbation-Based Computing

**Key idea.** Perturbation-based computational models are ideal for implementing complex non-linear filters (operators) associated with real-time information processing. The key idea is to perform a non-linear projection of the input stream into a high dimensional space using a complex dynamical system. If the pool of dynamics capturing information about current and past stimuli is sufficiently rich, *any* desired non-linear filtering task’s output(s) can be derived, or ‘composed’ from it. Below we develop this basic idea in a more rigorous manner.

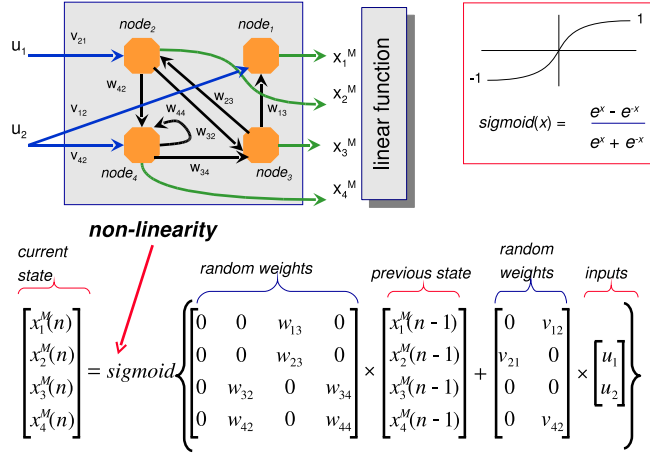
**Perturbation-based machines.** Fig. 1 symbolically depicts a perturbation-based machine  $M$ . As can be seen, it maps an input function  $u(\cdot)$  to an output function  $y(\cdot)$ , relying on two key components: a high dimensional dynamical system, implementing the machine’s *computational core*  $D^M$ , and an *output stage*  $f^M$ . The key premise underlying perturbation-based computing is that, by using computational cores realized by sufficiently complex, even random, dynamical

systems, one can essentially project inputs over a sufficiently large family of basis operators for any given set applications and desired approximation level [13]. A machine's  $D^M$  is thus a dynamical system realizing a very large pool of candidate operators, while the above mentioned  $D^M$  denotes a specific set of basis operators required for a given approximation. As such, the *same* computational core  $D^M$  can be used in realizing various tasks. The output stage is the *task dependent* part of this machine, playing the role of both selecting and composing the 'relevant' basis operators through a memoryless function.

As shown in Fig. 1, the computational core  $D^M$  generates an internal state  $x^M(t)$ , corresponding to a causal response to the input  $u$ . This is a non-linear projection of the input stream on a high dimensional space, generated by exciting the dynamical system associated with  $D^M$ . Note that *no stable internal states* are required in the computational core, it suffices to generate a sufficiently rich pool of transient dynamics. As such, one can say this computational model is *non-Turing* – a key departure from conventional computational models. The output stage  $f^M$  maps the internal transient state to the desired output.

***Universal approximative power and target applications.*** Based on Boyd and Chua's fundamental result [15], Mass established that perturbation-based machines have universal computing power – that is, machines operating 'natively' under this computational model can approximate *arbitrarily closely* any time invariant fading memory operator [13]. Still, although Boyd's result tells us that the number of basis operators required by any such approximation is *finite*, it says nothing about how many such operators may be required in each case. If very high precision is required, the number of operators may be very high for certain tasks. The proposed computational model is inherently based on approximation. Therefore it is not expected to operate without errors, with perhaps the exception of approximation of very simple functions/operators. Although for some tasks this will be unacceptable, for others this presents an opportunity to tradeoff error rate against other costs, e.g., manufacturing cost, power consumption etc. An example of such tasks would be those involving real-time searches for opportunities, e.g., block matches across frames in video compression. If we miss an opportunity this will not cause algorithm failure, instead it results in a temporarily lower compression rate. Another class of applications involves systems with feedback, where such small/rare errors can be compensated subsequently through feedback overall having a negligible effect. More generally the aim is not to achieve high precision, but rather simplicity and universality. Blocks having moderate reliability can be bootstrapped to construct more complex and/or reliable operators, e.g., through averaging or other forms of aggregation [16] or specific mechanisms akin to the way complex logic functions are constructed from elementary logic gates (e.g. NAND, NOR, etc).

***Modeling perturbation-based machines.*** In practice, the dynamical system comprising the machine's computational core,  $D^M$ , can, for example, be realized by a complex (randomly generated) recurrent network of non-linear operator nodes [13, 14]. In fact, given the rich pool of dynamics generated by such networks, the machine's task dependent output function can, in general, be quite



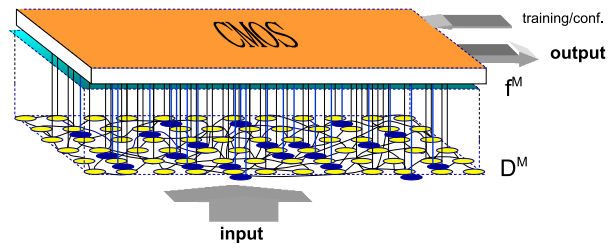
**Fig. 2.** Computational core and output stage of a small discrete-time perturbation-based machine.

simple, e.g., *linear*. Accordingly, for all experiments reported in this paper, only linear readout maps were considered. In this case one can exploit standard linear techniques to determine appropriate output functions: linear regression for tasks with real-valued outputs, and/or linear classification for discrete outputs [17].

Relying on the formal definition and broad principles given above, one can still build many variants of a perturbation-based machine, e.g., operating in discrete or continuous time, relying on different types of non-linear nodes, etc. For illustrative purposes, Fig. 2 shows an instance of a perturbation-based machine operating in discrete time, where each of the core’s nodes apply a sigmoid function (scaled to the range  $[-1, 1]$ ) to a weighted sum of their inputs. Due to space limitations, we depict a very simple computational core comprised of a recurrent network with only 4 nodes. (For the actual experiments reported later in the paper, much larger sparse incidence matrices defining the connections and weights of the corresponding recurrent networks were randomly generated.<sup>1</sup>) As shown in Fig. 2, the next state of the computational core (i.e., of each of its nodes  $x_i^M(n)$ ) is computed based on the core’s previous state and the current inputs. In turn, the task dependent linear readout function at the machine’s output stage is defined by assigning a corresponding weight (in the picture, denoted  $k_i$  for node  $n_i$ ) to each of the core nodes.

**Implementing perturbation-based machine.** Note that the precise internal dynamics of the core network need not have a specific form, as long as the core projects the input signals on a sufficiently rich set of basis functions.

<sup>1</sup> Note that in order to avoid chaotic behavior, such randomly generated sparse matrices were then scaled, so that the absolute magnitude of the maximum eigenvalue is 0.95, see details in [17].



**Fig. 3.** Hybrid eNano-CMOS configurable platform for perturbation-based computing.

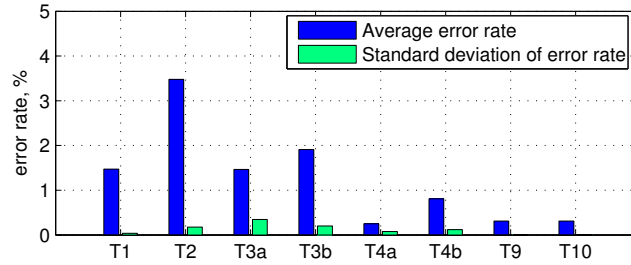
As such the precise implementation of the core used in the experimental results presented in this paper, should ideally be irrelevant, see [18] for details. Our results were generated by simulating cores whose dynamics follow the discrete time model illustrated in Fig. 2. This model is very similar to the ECHO model proposed in [14], except that in our case: (1) node-to-node connections within the computational core were generated introducing a strong bias towards local connections, so as to reflect practicality concerns<sup>2</sup>; and (2) there is no feedback loop projecting the output back to the computational core, as in the standard ECHO model. This machine is somewhat abstract, since the sigmoidal non-linearities could be complex to realize in practice. Yet as discussed further below, the key idea here is that analog/digital dynamical networks that make up a computational platform’s core, would be realized using whatever nanoelectronic devices are available, and need not be precisely engineered.

### 3 Design Principles for Perturbation Based Computing

**Hybrid platform.** We begin this section by first proposing a hybrid eNano-CMOS platform as a representative realization of perturbation-based machines. The machine’s computational core is implemented on an emerging nanoelectronic fabric while CMOS is used to implement the simple (e.g., linear) read out function at the output stage and support the machine configuration/training process. Fig. 3 shows an abstract view of such a platform, with the key basis operators in the pool highlighted in bold. Clearly, this platform can directly leverage the formidable densities achieved by nanotechnologies to create computational cores of essentially arbitrary size. At the same time the more reliable CMOS layer allows to reliably configure (train) the output readouts to properly approximate the desired function. We discuss five underlying design principles next.

**Design Principle 1: Defect-tolerance, randomly assembled cores and configuration/training.** Our first principle is that one need only ensure that the core and readout connectivity are sufficiently ‘rich’ to achieve the

<sup>2</sup> Concretely, when generating a machine core, we embed its corresponding nodes on the integer points of a 2D or 3D grid. Then, as done in [13], relying on the resulting Euclidian distances between core nodes, we randomly choose connections between them, using a probability low favoring shorter/local connections.



**Fig. 4.** Average and standard deviation of task error rates obtained for a pool of machines with randomly generated computational cores of a target size.

desired approximation after training. In other words the designer needs only control the size and statistics of the core network without precisely specifying its topology. Manufacturing defects and heterogeneity in the non-linearities in the network thus become part of its intrinsic randomness of the network. The experiments shown in Fig. 4 empirically support this point. These results show the task error rates achieved, across a suite of benchmark tasks, T1 -T10 realized on randomly generated computational cores of a similar (sufficiently large) size. As can be seen there is negligible variation in their ability to perform the task, i.e., have essentially the same computational power – as assessed based on task error rates. Details on the experiment’s on benchmark tasks, including channel equalization, voice recognition, visual motion prediction, and robotics as well as our experimental methodology can be found in [18].

This flexibility comes at the cost of performing a configuration/training step for each chip – which is indeed a costly requirement. Yet it can be viewed as ‘similar’ to the overheads associated with typical defect tolerance approaches. Indeed the typical requirements in the latter are to detect, i.e., map out, defects for each chip and then re-synthesize the function to avoid defects. Defect mapping is typically done using test patterns that are either obtained/generated off chip or stored on chip. Re-synthesis involves reprogramming the function around the defects on the chip. In our case rather than defect mapping and re-synthesis steps we require a training step. Such training will involve access to input-output pairs that can also be provided either off-chip or on-chip. A comparison of the cost of mapping an re-synthesis vs training is premature. Another potential disadvantage is possibility of excessive power consumption by random core in contrast to precisely controlled core. However reduced control requirements also open opportunities to make these random cores very cheap especially in terms of power consumption.

**Design Principle 2: Fault-tolerance through unstructured redundancy.** Our second principle is that fault-tolerance can also be partially achieved by appropriately defining the statistics and size of the core. Intuitively, even if randomly assembled, a large dynamical network should incorporate sufficient redundancy to allow the readout layer to average out internal noise/soft errors. We refer to this as unstructured redundancy in the core, as it need not be

explicitly designed, e.g., as would be the case with triple-module-redundancy. Instead the designer need only decide on a sufficiently large core to address soft faults and/or internal performance variability, e.g., due to coupling etc. Extensive results in [18] support the notion that larger cores are indeed more robust to transient errors. An obvious advantage of this approach is reduction in design cost in comparison to structured redundancy. A disadvantage this comes at a cost, bigger cores will consume more power.

***Design Principle 3: Complete core sharing.*** Note that aside from general considerations on size and network statistics detailed core characteristics are task independent. Thus several different tasks that share the same input can in principle share its projection on the same core. For example word recognition and speaker identification tasks for the same speech input could share the same core. Such complete and parallel sharing of resources has the potential to substantially reduce overall system cost in terms of both area and power. Note however that the readout layer can not be easily shared across tasks, which may lead to a scalability problem if a core is to be shared among a large number of tasks.

***Design Principle 4: Weakly interconnected networks and spatial decomposition.*** A potential problem with scaling to large cores is a scalability problem if random interconnections among nodes are necessary see e.g., [12]. We propose a fourth design principle towards overcoming this problem. The idea is that to introduce some hierarchy by only weakly interconnecting smaller cores. This allows one to control the interconnect costs as the size of the cores increase. Moreover this seems a natural way to randomly assemble cores, i.e., one where the primary form of connectivity is local. More generally one can imagine designs that leverage a large number of relatively small cores which serve as building blocks to create bigger cores as needed. Again extensive simulations in [18] supports this principle.

***Design Principle 5: Nearly decomposable core dynamics.*** The last design principle we propose relates to decomposition in terms of temporal dynamics. The idea is that some applications are driven by (possibly coupled) dynamics at different time scales, which a designer might recognize and incorporate into his core design. For example a core design might include weakly interconnected cores operating at different speeds. One can imagine, creating cores with different response times to input signals, through some form of doping and/or processing. For applications exhibiting dependencies on multiple time scales such decompositions are very effective at reducing complexity – see e.g., [18]. Furthermore purposefully combining fast and slow cores may present further advantages towards reducing power consumption. Note that the principle here is not perform careful core design, but simply define some large scale characteristics for connectivity and dynamics of its constituent subnetworks.

It should be clear at this point that our design principles in turn point to some of the technological challenges that one might explore in looking to leverage nanotechnologies for realizing perturbation based computing. One can for exam-

ple imagine work towards controlling the connectivity statistics and character of dynamics and nonlinearities through chemical means and random assembly.

## 4 Contrast to previous work

***Contrast to other computational models.*** As mentioned above, perturbation-based computing relies on transient internal states, and is therefore not a Turing model. That is, unlike Turing machines, or even less powerful standard finite state machines, the only stable state of a perturbation-based machine’s computational core is in general the ‘rest’ state, and the machine has no ‘permanent’ memory. These characteristics make this computational model unique. Furthermore, although perturbation-based computational models are likely to use recurrent networks to realize their computational core, they are also fundamentally different from traditional recurrent neural networks (RNNs). Indeed, after training, RNNs operate essentially as deterministic FSMs. That is, they exhibit a finite number of attractor states, which encode their possible outputs. A key challenge for RNNs is designing system dynamics so as to create suitable stable/attractor (or ‘low energy’) states, so that a network subject to specific inputs eventually converges to the correct stable states. Such convergence is hard to achieve, since dynamic systems may easily become unstable – thus, most research in the field deals with very simple special cases or network topologies [17]. By contrast, the operation of perturbation-based machines does not center on designing/controlling dynamics, but rather draws on the rich transient dynamics of complex (and possibly randomly assembled) systems, operating under substantial structural uncertainty.

***Previous research on perturbation-based computing.*** As mentioned earlier, perturbation-based computing was independently proposed by two research groups [13, 14], both of which have furthered this area, but have fundamentally different objectives than ours. The main objective in [13] was to model *biological* neural circuits and understand the operating principles of such biological systems. Accordingly, ensuring biological plausibility (e.g., for the network nodes), and successfully mimicking the behavior of actual neocortex circuits (e.g., in terms of possible information encoding), were key drivers for their research [13, 19, 20]. These issues are not germane to our research, and in fact obscure our core objectives. The work of [14] was driven by the desire to develop practical engineering techniques for training (artificial) recurrent neural networks, to be used in control applications. Their main objective was to circumvent the need to control (design) complex network dynamics – an exceedingly hard problem [17]. Note, however, that this line of work assumes that such networks/models are to be directly programmed, using Turing complete languages, on conventional general purpose computers. By contrast, our focus and contributions are directed towards realizing machines that operate directly under this computational model, rather than emulating or simulating it using Turing complete machines/languages. In particular, we are interested in assessing the suitability of perturbation-based computational models for nanotechnologies characterized by

high defect density and high performance variability – a major research challenge posed to the computer science and computer engineering communities.

## 5 Conclusion

Much current research towards realizing ”nanocomputing” is based on assumption that computing will be done in similar fashion as it is today, but with many more more devices that are less reliable. However, an alternative view is that future nanocomputing systems could operate differently for special application classes. This paper explores one such alternative. In our case we focused on tasks which require approximating non-linear operators over time varying inputs which we believe to be relevant to future embedded systems. We have pointed out a variety of interesting characteristics, such as the use of randomly assembled nano networks cores. This approach is consistent with circumventing with what appears to be an intrinsic difficulty and thus costly requirement of precisely controlling the manufacturing of nano systems. So the question is to what degree can one give up on precise control over manufacturing and still devise useful computation systems? In this paper we presented one possible approach but there may be others (perhaps more general) models.

## 6 Acknowledgments

This work is supported by the Gigascale Systems Research Center (GSRC), under the ‘Alternative’ Theme.

## References

1. V. Zhirnov and D. Herr, “New frontiers: Self-assembly and nanoelectronics,” *IEEE Computer*, vol. 34, no. 1, pp. 34–43, January 2001.
2. J. Heath, “Wires, switches, and wiring: A route toward a chemically assembled electronic nanocomputer,” *Pure and Appl. Chem.*, vol. 72, no. 11, 2000.
3. Y. Huang, X. Duan, Y. Cui, L. J. Lauhon, K. Kim, and C. Lieber, “Logic gates and computation from assembled nanowire building blocks,” *Science*, vol. 294, no. 5545, pp. 1313–1317, 2001.
4. S. C. Goldstein and M. Buidu, “Nanofabrics: Spatial computing using molecular electronics,” in *Proc. International Symposium on Computer Architecture (ISCA)*, July 2001, pp. 178–191.
5. A. DeHon, “Array-based architecture for FET-based nanoscale electronics,” *IEEE Trans. Nanotechnology*, vol. 2, no. 1, pp. 23–32, 2003.
6. M. F. Jacome, C. He, G. de Veciana, and S. Bijansky, “Defect tolerant probabilistic design paradigm for nanotechnologies,” in *Proc. IEEE/ACM Design Automation Conference (DAC)*, 2004, pp. 596–601.
7. A. DeHon, S. G. Goldstein, P. J. Kuekes, and P. Lincoln, “Non-photolithographic nanoscale memory density prospects,” *IEEE Trans. Nanotechnology*, vol. 4, no. 2, pp. 215–228, 2005.

8. G. Bourianoff, "The future of nanocomputing," *Computer Magazine*, pp. 44–49, Aug. 2003.
9. J. R. Heath, P. J. Kuekes, G. S. Snider, and R. S. Williams, "A defect-tolerant computer architecture: Opportunities for nanotechnology," *Science*, vol. 280, pp. 1716–21, June 1998.
10. M. Mishra and S. C. Goldstein, "Defect tolerance at the end of the roadmap," in *Proc. International Test Conference (ITC '03)*, 2003.
11. "Sematech. international technology roadmap for semiconductors - 2004 update on emerging research devices." <http://www.itrs.net/Common/2004Update/2004Update.htm>.
12. A. Zykov and G. de Veciana, "Exploring density-reliability tradeoffs on nanoscale substrates: When do smaller less reliable devices make sense?" *The 23rd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT'08)*, 2008.
13. W. Maass, T. Natschlag, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
14. H. Jaeger, "The echo state approach to analysing and training recurrent neural networks," *GMD-Report 148*, German National Research Institute for Computer Science, 2001.
15. S. Boyd and L. Chua, "Fading memory and the problem of approximating nonlinear operators with volterra series," *IEEE Trans. on Circuits and Systems*, vol. 32, pp. 1150–1161, 1985.
16. G. V. Varatkar, S. Narayanan, N. Shanbhag, and D. L. Jones, "Sensor network-on-chip," in *International Symposium on SOC*, November 2007.
17. H. Jaeger, "Tutorial on training recurrent neural networks, covering bppt, rtrl, ekf and the "echo state network" approach," *GMD-Report 159*, German National Research Institute for Computer Science, 2002.
18. A. Zykov, M. Jacome, and G. de Veciana, "Perturbation-based computing for next-generation embedded IT targeted at emerging nanoelectronics," *Submitted for publication*, 2008.
19. W. Maass and H. Markram, "Theory of the computational function of microcircuit dynamics," in *The Interface between Neurons and Global Brain Function*, ser. Dahlem Workshop Report, S. Grillner and A. Graybiel, Eds. MIT Press, 2005, vol. 93.
20. W. Maass, T. Natschlag, and H. Markram, *Computational models for generic cortical microcircuits*. Chapman & Hall/CRC, Boca Raton, 2004, ch. 18, pp. 575–605.