Biologically Inspired Computing Algorithms: Relevance and Implications for Research Technologies

Scott McCaulay

Indiana University

PTI Technical Report PTI-TR12-003

1 February 2012

Citation:

McCaulay, S. "Biologically Inspired Computing Algorithms:
Relevance and Implications for Research Technologies," Indiana University,
Bloomington, IN. PTI Technical Report PTI-TR12-003, Feb 2012.

Table of Contents

Table of Tables

1. Abstract

The evolution of the cyberinfrastructure which supports scientific research in the US cannot be explained solely by observation of advances in computing hardware. Rather it represents a complex interaction among many factors, which include advances in hardware, but also include new opportunities in deployment options, new tools and protocols and advances in programming tools and techniques. It takes an unusual confluence of factors to prompt a significant number of user communities to simultaneously consider abandoning an established set of codes in favor of a potentially risky new development project. Many of the factors involved in these decisions are well understood and frequently discussed. The evolution of processor speeds, new types of processors, new deployment methods utilizing distributed or virtualized resources, the general movement in the focus of research technology away from intense computation and toward massive data; these are among the most common areas of discussion in HPC circles. Much less commonly discussed, and the topic of this paper, are the types of algorithms used in research applications. Specifically, we will look at the most frequently used HPC algorithms, along with a collection of bio-inspired computing algorithms which so far are not in routine use in HPC. Our questions are whether these bio-inspired algorithms, in combination with ongoing sweeping changes in hardware, deployment and data-centricity, will contribute to a potential solution which will offer a sufficiently attractive opportunity to improve the efficiency and flexibility of scientific codes. We will also consider, should these algorithms become a part of the next wave of scientific code development, what will be the implications for the kind of infrastructure that will be in demand to support the new codes.

Keywords: Algorithms, bio-inspired computing, research cyberinfrastructure.

2. Introduction

Many of the staples of today's research computing environment date back decades, even generations in their origin. This is not a condemnation of the environment, rather an acknowledgement that reliable and effective paradigms have been established which suit the needs of this community extremely well and have been adapted and used successfully over many years. The commodity cluster model for hardware that continues to dominate the offerings and usage of computing cycles for scientific research today was introduced conceptually in the mid-1990s and became widely adopted within a few years [1]. Nearly simultaneously, a diverse group of software developers agreed to combine their previously somewhat splintered efforts to produce a common message passing interface [2]. These and other developments, including funding for software development through the NSF's HPCC initiative, contributed to a sea change in the scientific computing environment. Many of the most commonly used scientific codes today were first initiated and developed in this time period, from the mid to late 1990s.

The creators of the NAMD code expressed that the driving forces behind their new development effort in 1996 included their desire to tailor their molecular dynamics work both to the then emerging cluster environment and the new MPI standard, and also to implement some concepts of modular development and deployment in C++ which would make the code more extensible and maintainable over time [3]. The WRF project for weather forecast modeling, launched in 1998, expressed some similar motivations [4]. Specifically, their previous codes had relied heavily on Cray-specific constructs, and they were interested in a new set of codes that would be more portable in a cluster environment using new standards such as MPI. They also mentioned goals of modularity, extensibility and maintainability, as well as new features available in Fortran 90 which had been released as an ANSI standard several years earlier in 1992. In these cases and others, it was a complex combination of factors that made the development effort worthwhile. The newly available commodity clusters were a big contributing factor, but the developers point to standardization of tools such as MPI, to the general acceptance of improved programming techniques, and even to the evolution of the programming languages themselves as additional motivating factors.

We see a number of revolutionary opportunities emerging and available today which could be applied to scientific computing. Most notably, the much ballyhooed availability of cloud computing does offer some promise to the scientific community [5]. Additionally it has been widely accepted for years that scientific computing is evolving away from the established dominance of a computationally intensive model toward a data-intensive "Fourth Paradigm" model [6]. Commonly implemented programming techniques continue to evolve and also provide new opportunities; modularity and extensibility have reached new levels within an increasingly on-demand and service-related delivery model. Despite the availability and high visibility of these new opportunities, we have not seen so far a mass exodus from the local physical cluster, high speed interconnect, message passing model which has been solidly established for now approaching twenty years.

There are clear reasons why the established paradigm remains so firmly rooted. Scientific communities have shown a tendency to retain the complex sets of codes that support their research over long periods of time with only incremental maintenance and enhancement. Communities develop not just a comfort level with the codes, but also a level of confidence. The critical importance of accuracy, reliability and reproducibility to these communities cannot be overstated, and as such the burden of extensive testing adds considerably to the already high cost of initiating a new development project. To entice a community to undertake such an effort, the potential gains in some area of performance or convenience have to be perceived as clearly outweighing some pretty significant costs.

The understandable inertia within the scientific code bases represents just one factor that favors the continuation of established methods within the scientific research cyberinfrastructure. Difficulty of development, familiarity and proven reliability of established codes, ease in replicating or building on prior research, all of these contribute to the perceived value of maintaining both an existing code base and the established environment in which it runs. There are also social factors that encourage continuity. There is a sort of "rite of passage" involved in learning not only the codes but the esoteric tools and environment that surround the usage of the codes. Making a successful effort to learn these complex techniques not only creates a bond within the research community, but it bonds that entire group to a larger community of HPC users. This social phenomenon contributes to and reinforces unfortunate negative stereotypes surrounding the HPC community in academic research, that it is an exclusive club with a dismissive attitude toward outsiders, and with a greater interest in maintaining and expanding their budgets and their inventory of ever more expensive and sophisticated toys than sincere interest in providing broad support for the academic research community.

Much has been made of challenges to the research cyberinfrastructure status quo which come from new deployment and delivery methods for computing resources. Virtualization and ubiquitous networking make it feasible to make cores and even clusters available on the fly and on demand. Considerably less has been said about the evolution of algorithms over the past decades. An emerging movement over the last decade which may eventually impact research computing in a big way has been the increasing popularity and broad application of algorithms that model biological processes. These bio-inspired computing methods have had an interesting relationship to the history of computing. Much of the work that led directly to the design and development of the earliest modern computers in the 1940s were built on biological models [7]. Despite their early dominant influence, bio-inspired algorithms such as neural networks did fall out of favor for some time. In the last decade, bio-inspired algorithms have found new life particularly in big data applications [8].

3. The Traditional HPC Algorithms

The definitive statement on the algorithms that enable scientific computing continues to be Phillip Colella's 2004 presentation on the "Seven Dwarfs" of high performance computing [9]. This was later expanded from seven to thirteen by Asanovic, Bodik et al. [10]. In both cases, the claim is that nearly all high performance computing done today in support of scientific research uses one or more of a very small set of algorithms. The Asanovic, Bodik et al. paper also provides detailed analysis of the inter-process communication patterns typical of each of these commonly used algorithms; these analyses are the basis for assumptions made in this paper about the relative communication requirements to support and enhance the performance of these algorithms.

One purpose of this paper is to review these traditional algorithms, alongside the bio-inspired algorithms, to explore which set of algorithms are better suited to the big data era of scientific computation, and also to explore whether a trend toward more widespread adoption of bio-inspired algorithms would result in user demand for different types of computational, storage and network resources. One thing that is clear from even a cursory review of these algorithms which have long formed the foundation of scientific research computing is that nearly all of these algorithms have a moderate to high dependence on inter-process communication. At the extreme, some of these algorithms require "all to all" communication in which the calculations within every process are dependent on information from every other process. Obviously applications built on such algorithms are the perfect candidates for the types of clusters which continue to dominate the research infrastructure today. In fact the resource providers and the application users together create a self-sustaining cycle in which the needs of the applications continue to demand the provision of the physically interconnected cluster, and the providers of these systems in turn show a preference for the applications which best justify the investment in that architecture. While these are valid computing needs in support of important science, the question remains, when will a widely accepted alternative to this model emerge which will serve a wider audience of scientific researchers, and which will also address other needs of scientific communities such as on-demand access to resources or better methods to deal with unprecedented quantities of data?

This paper will make no attempt to provide any in depth description of these algorithms; that level of detail is well beyond our scope. Our only intention is to look very broadly at what are considered the most significant algorithms underlying scientific computing today, to consider the applicability of these algorithms to the type of tightly coupled clusters still dominating our research cyberinfrastructure, and consider the combination of algorithms and hardware as a self-influencing and self-sustaining system.

Colella's original seven dwarfs included the following algorithms:

Algorithm	Communication Needs
Dense Linear Algebra	Moderate
Sparse Linear Algebra	Moderate
Spectral Methods	High
N-Body Methods	High
Structured Grids	Moderate
Unstructured Grids	Moderate
Monte Carlo	None

Table 1. Colella's Seven Dwarfs.

3.1. Dense Linear Algebra

These algorithms are typified by packages such as BLAS (Basic Liner Algebra Subprograms) [11]. Very generally, these algorithms perform mathematical operations on vectors or matrices of values, and as implemented for high performance environments they are reliant on message passing among processes, and so are dependent on a high speed interconnect for performance. There are many implementations of these types of algorithms, typically in Fortran or C. Vendors frequently provide libraries to optimize these algorithms for their hardware. Often new hardware environments are crippled by the lack of readily available algorithms. In recent years, the use of GPUs for scientific programming has become more realistic with the availability of parallel linear algebra libraries from AMD and NVIDIA for OpenCL [12] and CUDA [13] respectively.

The LINPACK and HPL benchmarks both provide measures of performance on dense linear algebra routines, using the DGEMM (Double-precision General Matrix Multiply) subroutine from the BLAS library to measure performance. These algorithms do rely on inter-process communication but fairly moderately, as there are many processes which do not need to communicate directly with each other.

3.2. *Sparse Linear Algebra*

Sparse linear algebra also deals with performing mathematical operations on vectors and/or matrices, but in this case the vectors or matrices may contain many empty values. Compression methods are employed to store and retrieve only the non-zero values which are relevant to the calculation. There are dozens of routines and solvers available for this type of processing; an example would be the SuperLU library [14] developed at the University of California at Berkeley.

The sparse linear algebra algorithms also have a moderate reliance on inter-process communications for performance, somewhat higher than the dense linear algebra algorithms.

3.3. *Spectral Methods*

Spectral methods deal with transformation of spatial or temporal data. Fast Fourier Transformation [15] is a widely used example of this type of algorithm. These types of methods are applicable to many types of scientific research, and are used in areas such as climate research, astrophysics and nanoscience. The PFASC [16] system, a similarity analysis and clustering code developed at Indiana University is an example of a research application using spectral methods.

These algorithms are extremely reliant on inter-process communication to optimize performance in a parallel environment. Some implementations require all-to-all communication in which every node communicates with every other node.

3.4. *N-Body Methods*

These methods in general deal with interactions among many points or bodies, hence the name. A typical application is modeling interaction among many particles. This includes models where the action of every particle is dependent on the state of all other particles in the model for each timestep. Though there are a number of different methods that make up this category, it is fair to say that in general, these algorithms have a very high need for inter-process communication to improve their performance.

Molecular dynamics applications are representative of the types of scientific codes built around these algorithms. N-Body problems are one set of scientific computing problem which are so pervasive and computationally intensive that specialized hardware has been built solely to address these problems [17].

3.5. *Structured Grids*

These algorithms model a multidimensional grid of values (anything from 2D up, but 2D and 3D are most common). This is another timestep type of algorithm in which all values in the grid are updated at each step. Unlike the all-to-all communications required in some N-Body and Spectral algorithms, in a structured grid, it is only the state of some immediate neighborhood that impacts the new value at each cell.

The Cactus framework and toolkit [18] is an example of a commonly used scientific code built around structured grids. These methods are moderately dependent on inter-process communications, but not at the all-to-all level.

3.6. *Unstructured Grids*

Unstructured grid problems deal with an irregular mesh or grid, where calculations to update each element are dependent on values of neighboring elements. While a structured grid can be characterized simply as a multidimensional array, an unstructured grid is complicated by the fact that elements are of irregular shape and size. Characteristics of the elements must be provided explicitly. The mesh structure itself may change over the course of the simulation.

This is yet another example of an algorithm in which cell values are updated through a sequence of time steps. As in the structured grid, each cell requires information about its neighbors in order to update, so these methods require a good deal of inter-process communication.

3.7. Monte Carlo Methods

Monte Carlo methods rely on a series of random simulations rather than an exact deterministic process. This was one of the earliest algorithms commonly used in scientific computing and it remains in use to this day. These algorithms were used on the Manhattan Project and in the early days of the Los Alamos National Laboratory [19]. They have enjoyed a long popularity not only in science but have also been used in commercial industry such as Telecommunications and Finance.

Monte Carlo algorithms are the only example among Colella's original seven which are NOT dependent on inter-process communication, each process runs independently. Monte Carlo problems are often referred to in the HPC community as embarrassingly parallel, a nomenclature that in itself speaks volumes about attitudes within that community. Among Colella's seven, these algorithms also share the most with the bio-inspired algorithms we will be looking at, in that they also are non-deterministic in nature and rely on a random progression to arrive at a solution.

3.8. Other Algorithms

Asanovic, Bodik et al. revisited Colella's seven dwarfs, considered a number of other commonly used algorithms, and compiled a list of thirteen significant algorithms. We will not look in detail at each of these, but we will comment on a few significant inclusions and exclusions.\

Algorithm	In Colella?
Dense Linear Algebra	Yes
Sparse Linear Algebra	Yes
Spectral Methods	Yes
N-Body Methods	Yes
Structured Grids	Yes
Unstructured Grids	Yes
MapReduce	No
Combinational Logic	No
Graph Traversal	No
Dynamic Programming	No
Back-Track, Branch+Bound	No
Graphical Models	No
Finite State Machine	No

Table 2. Asanovic, Bodik et al. Thirteen Dwarfs.

Interestingly, the new set of algorithms, unlike the original seven, includes some algorithms which are associated with very different types of processing and even with different types of architecture. Some of the newly added methods are associated with big data, some are associated with cloud computing, and some of the methods mentioned by the authors as considered for inclusion are bona fide bio-inspired algorithms. This could be seen as, even in the short period of time between the two papers, some movement toward accepting a broader definition of what constitutes scientific computing.

The new list eliminates Monte Carlo methods and specifically replaces them with MapReduce [20]. This replacement was conscious, as the authors consider MapReduce to be a variation of the Monte Carlo method. MapReduce is another embarrassingly parallel algorithm with no dependence on inter-process communication. It grew out of the Information Retrieval discipline, but has found general application for data mining.

Combinational Logic and Graph Traversal, two more algorithms added to the new list, are also associated with big data applications. Construct Graphical Model, Finite State Machines, Dynamical Programming and Back-Track/Branch & Bound all include some methods associated with artificial intelligence. In general, the new set of algorithms has less emphasis on deterministic outcome, more emphasis on data, and less dependence on inter-process communication.

4. Biologically Inspired Computing

The early history of computation as we know it today is intricately intertwined with the history of biologically inspired computing. John Von Neumann, who really articulated the concept of the digital computer as we think of it, was also a pioneer in the areas of artificial life and artificial intelligence, having developed among other things the concepts of game theory, artificial life and the cellular automata [21].

Much of the groundbreaking work that led to the development of the earliest modern computers came out of the Macy Conferences on cybernetics, which was an interdisciplinary effort originally intended to develop a new science of human mind [22]. Several important new fields of study did originate from these events, including Cognitive Science and System Theory. The attendees covered many fields; they included mathematicians, psychologists, logicians, anthropologists and sociologists. While much of this work built on earlier ideas, it was the work of the Cybernetics group through these conferences that led directly to the creation of the EDVAC, which established the model for modern computing. The renewed interest today in biologically inspired computing represents more of a return to appreciation of concepts prevalent and popular in the early days of computing than a new phenomenon.

After being the inspiration for the development of electronic computing, bio-inspired computing fell out of favor for some time before its recent resurgence. There are a number of reasons why it lost its initial popularity. With regard to scientific computing, these reasons are easy to understand in relation to the computationally intensive algorithms which did gain popularity. Scientific computing has shown a strong historical preference for deterministic algorithms, algorithms which can be depended on to run the same way and produce reliable results. Most of the bio-inspired computing algorithms have a strong random component to them. Using genetic algorithms, swarming intelligence, neural networks and so on, each run may produce a different solution. Even if the ultimate optimal solution returned is the same, a different path will be followed to arrive at that solution.

In many fields of study, this deterministic predictability is not important and in many cases, the many possible solutions are an important part of the research. We can observe examples of computation at massive scales occurring in nature, carried out in ways that are decidedly non-deterministic [23], and these algorithms can model that behavior. Bio algorithms have been used to model and analyze very large scale complex systems, such as market behavior, online activity, traffic patterns and so on. Because of the non-deterministic nature of the algorithms, researchers can run many models and analyze results to find likely real world outcomes. . Unlike Monte Carlo methods, bio-inspired algorithms are not simply a series of random parameters, but rather are guided efficiently toward a solution, so that in a problem with a staggering number of possible solutions, most can be eliminated without being explicitly tested.

The fundamental difference in the way in which bio algorithms can be parallelized, relative to the traditional HPC algorithms, is critical to an understanding of how they could be effectively deployed. The bio algorithms would be classified as embarrassingly parallel in that they can run many independent processes with only minimal need to communicate among them. Some of the traditional algorithms rely on all-to-all communication for parallelization, which places a lot of strain on scalability.

Reliance on the communication-intensive traditional algorithms has resulted in an odd sort of symbiosis between the providers of HPC resources in support of scientific research and their users. One of the most significant metrics the resource providers use to measure their success is their ability to provide systems that score highly on benchmarks such as Linpack which emphasize high-speed inter-process communication. By relying on algorithms which required all-to-all process communication, users drove the demand for these relatively expensive and highly specialized systems. Resource providers in turn value users who best justify the architecture. The resource providers and users together became a tightly coupled and highly insular community. While only a very small part of all contemporary scientific research is supported by the use of large scale tightly-coupled computing clusters, it has become common to equate that computing paradigm with scientific computing as though they are a single and indivisible concept.

Applications based on the bio algorithms could potentially run under very flexible and constantly changing conditions. These algorithms could be ideal for taking advantage of an environment in which the availability of resources may be heterogeneous, dynamic and unpredictable. Such applications could work effectively running a large number of jobs of extremely short duration, and would not even require a fixed number of processes running simultaneously. An extremely large scale system of virtual resources, made available on demand could provide an interesting opportunity to support many types of scientific research which are underserved by the HPC-focused models in common usage today. The enormous growth in hardware resources in support of enterprise activities at contemporary major universities, and the search for opportunities to utilize surplus and idle resources in support of other aspects of the university mission, adds another element to the environment in which a major paradigm change is possible in the near future.

4.1. Evolutionary Algorithms

Genetic algorithms [24] are the most familiar and commonly used evolutionary algorithms. Any problem that has a solution which can be defined as a set of strings or numbers can be approached using a genetic algorithm. They can be very effective and efficient method for solving nonlinear problems.

A genetic algorithm consists of a population of potential solution sets. Each potential solution, stored as a vector of values, is seen as the genotype of an organism in the population. An original population is generated at random. Subsequent populations are evolved from combinations of existing solutions. A fitness function determines each solution's success. More successful solutions are more likely to be represented in future generations. Over time, the population evolves in such a way that it contains more solutions which score higher against the fitness function.

To create a new population member, some function chooses two parents, based in some way on their fitness success. Some implementations include concepts of geographic proximity, so that potential parents need to be neighbors. There are some variations on how the genetic material is passed to offspring. It could be random for each position, or there could be one or more crossover point(s) chosen at random. Mutations are an essential part of the algorithm. Without mutations, the algorithm risks premature convergence on a suboptimal solution. Diversifying the solution set through mutations broadens the search space and keeps the search open to better solutions that may not have been present in the original random population.

Two points about genetic algorithms are important to consider in relation to the types of resources required to support an implementation. First, the computationally intensive portion of the algorithm is the application of the fitness function to each potential solution. This calculation is completely independent of any other member of the population and requires no outside communication. Second, the overall evolution of the population does not require a fixed population over time or any synchronization of the introduction of new generations or new population members. Though it is common to implement a genetic algorithm with a fixed population and synchronous generational reproduction cycle, these are just conveniences and not a requirement for the effective operation of the algorithm. Obviously the natural processes which inspire the algorithm are not so rigid about population size or reproduction cycles.

A genetic algorithm could make very effective use of a high throughput environment which provides cycles on demand. New population members could be generated on the fly based on available resources. Since the bulk of the computation is in the fitness function, very little stored information is required to create a new entity, just the fitness scores and genotype vectors of the parents. A scientific application developed around such an algorithm, would be largely indifferent to the small differences in performance between a physical and virtual cluster, and would not be dependent for its performance on high speed inter-process communications. In fact, the high availability of on-demand cycles would be the most important requirement for efficient performance of such an application, and that on-demand availability is precisely what cannot be readily provided by our traditional physical clusters.

4.2. Swarm Intelligence Algorithms

Swarm intelligence algorithms provide another alternative to search for optimal solutions for nonlinear functions. These algorithms are rooted in artificial life, and attempt to emulate behavior such as bird flocking and fish schooling [25]. There is also a similarity to genetic algorithms, in that each potential solution is conceived as a unique entity. In this case, rather than the elements of the solution being perceived as genetic material passed on to future generations of solutions which favor better solutions, the solutions are seen as a flock pursuing a goal and the elements that make up the solution represent that solution's position in a multidimensional solution space. To pursue the optimal solution, each entity positions itself relative to the flock member closest to the goal and relative to its own historical best position.

It is worth noting that one of the authors of the particle swarm algorithm is Russ Eberhart, who is the Chair of the Department of Electrical and Computer Engineering at IUPUI. The algorithm itself is very simple, and yet it can search a solution space very effectively. Each member of the flock is a potential solution, an array of values, which represents both its solution and its position in multidimensional space. Each member of the flock, in addition to its solution values, maintains its best values against a fitness function. At each cycle, each member calculates its distance from both its previous best position and the all-time best position of any flock member. Based on a set of parameters, it moves closer to these more optimal positions. If graphed, the combined behavior of all elements emulates very closely the movement of a flock of birds or a school of fish. Over time the flock closes in on the optimal solution.

The swarming algorithms have some characteristics in common with the evolutionary algorithms in that the computationally intensive part of the algorithm is the application of the fitness function to a potential solution, which is independent of any need for inter-process communication. Although in this algorithm, it is expected that the flock will retain a constant size, it is not required that all members move simultaneously or that all members have over time the same number of moves. And each flock member does not require a persistent dedicated resource to run, each cycle could be a separate job of very short duration, which would just have to store its ending state. Like the evolutionary algorithms, swarm intelligence algorithms could take good advantage of resources which were loosely coupled and sporadically available.

4.3. Other Bio-Inspired Algorithms

One interesting application of bio-inspired algorithms is genetic programming, in which computer programs are actually developed in an evolutionary fashion [26]. Using genetic programming, a population of programs can be generated at random by selecting from a pool of available functions. Programs grow and evolve based on their performance against a fitness function, using a tree structure in which subprograms can be inserted at random nodes into an evolving program. There is no obvious practical application for this concept in scientific computing today, but it's an exciting concept that could conceivably be useful at some point.

Neural networks have a special place in the history of computation, and they are still in common use today. But machine learning techniques in general do not fit the embarrassingly parallel model, and other machine learning techniques not based on a biological model are more commonly used in large scale scientific applications. For example, Support Vector Machines [27] use a dense linear algebra algorithm for machine learning.

While there has been some very interesting work done with L-Systems, Cellular Automata and other bio-inspired methods, the applicability of these and other bio-inspired algorithms to scientific computing is questionable at this point.

5. Challenges and Opportunities for Indiana University

It seems inevitable that we are approaching an era of rich opportunity for development of new codes for scientific applications, similar to the period observed in the late 1990s. It's not obvious exactly when this will occur, but the quantity and scale of the opportunities seem to be approaching a tipping point.

Indiana University has an unusually favorable environment to participate in the development of next generation scientific codes. The Complex Systems group at the School of Informatics provides considerable expertise on the practical application of bio-inspired computing. The central IT function, the University Information Technology Services (UITS), has become well established in large scale, multi-institution application development projects such as Sakai and Kuali. The Pervasive Technology Institute (PTI) at Indiana University also has a strong background and history of successful development projects in support of the national cyberinfrastructure. PTI is also the home institution of the FutureGrid distributed testbed. Testing the development and deployment of new types of scientific applications would have outstanding resources available through FutureGrid.

With these strengths, along with successful partnerships with communities of users, and a history of large scale development successes, Indiana University is in a position to be a leader in establishing a feasible new paradigm to support scientific computing in a broader and more cost effective way.

6. Summary

While not every scientific problem lends itself to solution using bio-inspired algorithms, it is evident that a growing number of communities are recognizing the value of these methods in support of their research. This paper is intended to emphasize a few main points. One point is that a common model has come to dominate the cyberinfrastructure of scientific research for the past ~20 years. That model is the combination of massively parallel clusters with high speed interconnects, algorithms which have high requirements for inter-process communication, and message passing protocols to accomplish that communication. These methods work extremely well for certain types of problems, and the environment and culture which have grown around them have shown remarkable staying power contrary to other areas of computing which may be dominated by faddism and rapid obsolescence.

The elements which make up this existing model are self-supporting and self-sustaining, not prone to evolution. New hardware systems are designed to meet the expectations and needs of existing applications and users. Preference is shown to provide cycles and support for applications that make good use of the specialized architecture. Over time the environment which was once novel becomes canonical and transcends accountability.

It's clear that a much larger potential audience of scientific researchers will never be served by the existing model. Even within the existing environment, studies have shown that many of the jobs run on our most capable supercomputers do not require or make use of their sophisticated capabilities [28]. It is possible that a larger and more diverse audience could be served by a radically different model based on a very different architecture and deployment methodology for computing resources. The bio-inspired algorithms presented here may prove to be an important part of the new architecture, by providing additional alternatives to the existing embarrassingly parallel methods and thus more opportunities to reduce the emphasis on inter-process communication and scale to much larger quantities of data. These bio-inspired algorithms are much more broadly applicable than MapReduce, and they have also been shown to outperform Monte Carlo methods. By making use of idle cycles of machines dedicated to other purposes, this new model could provide computing resources not only to support a broader array of scientific research but also at a much lower cost per cycle than the existing model.

The biggest question is, given the economic and social forces that support the existing model, how can that cycle be broken in any meaningful way? Three main paths forward are evident.

1) Proactively fund or support development of next generation applications which can model and showcase the effective use of already available virtual and on-demand resources.
2) Change the metrics by which HPC resource providers are measured. Rather than being rewarded for resource saturation, providers should be rewarded for:
    a. minimizing their queue wait times
    b. working with users to improve the efficiency of their codes
    c. encouraging resource-intensive users to utilize funded centers
3) Carefully consider sources of funding for local clusters that are geared to serve a small, specialized audience. Encourage such users to use appropriate national resources, and encourage local providers to provide more generalized and virtual resources, and to find ways to make idle enterprise resources available to research users.

7. References

1. Reschke, C., Sterling, T., Ridge, D., Savarese, D., Becker, D., Merkey, P. “A Design Study of Alternative Network Topologies for the Beowulf Parallel Workstation”, Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing. 1996.
2. Dongarra, J. and Walker, D. “MPI: A Standard Message Passing Interface”, Supercomputer 12(1):56–68, January 1996.
3. Nelson, M., Humphrey, W., Gursoy, A., Dalke, A., Kalé, L., Skeel, R.D. and Schulten, K. “NAMD-A parallel, object-oriented molecular dynamics program”, International Journal of Supercomputer Applications and High Performance Computing, 10:251-268. 1996.
4. Michalakes, J., Dudhia, J., Gill, D., Klemp, D., and Skamarock, W. “Design of a Next-Generation Regional Weather Research and Forecast Model”, Towards Teracomputing, World Scientific, River Edge, New Jersey, pp. 117-124. 1998.
5. Barga, R., Gannon, D. and Reed, D. “The Client and the Cloud: Democratizing Research Computing”, in IEEE Internet Computing, IEEE Computer Society. 2011.
6. Szalay, A. & Gray, J. “2020 Computing: Science in an Exponential World”, Nature 440, 413-414 (23 March 2006)
7. McCulloch, W. S. and Pitts, W. "A Logical Calculus of the Ideas Immanent in Nervous Activity." Bulletin of Mathematical Biophysics, 5, 115-133. 1943.
8. Engelbrecht, A. P. “Fundamentals of Computational Swarm Intelligence”, Wiley. 2005.
9. Colella, Phillip. “Defining software requirements for scientific computing.” Presentation. 2004.
10. Asanovic, Bodik et al. “The Landscape of Parallel Computing Research: A View from Berkeley.” Technical Report No. UCB/EECS-2006-183. 2006.
11. Blackford et al. “ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance” Computer Physics Communications, Vol. 97. 1996.
12. “ViennaCL: Linear Algebra on GPUs using OpenCL”, Retrieved January 23, 2012, from http://gpgpu.org/2010/06/15/viennacl
13. “NVIDIA CUDA Basic Linear Algebra Subroutines (cuBLAS)”, Retrieved January 23, 2012, from http://developer.nvidia.com/cublas
14. Demmel, J., Eisenstat, S., Gilbert, J., Li, X. and Liu, J. “A Supernodal Approach to Sparse Partial Pivoting” SIAM Journal on Matrix Analysis and Applications, vol. 20, no. 3, pp. 720–755. 1999.
15. Cooley, J. W. and Tukey, J.W. "An Algorithm for the Machine Calculation of Complex Fourier Series," Math. Comput. 19, 297–301. 1965.
16. McCaulay, D. S. and Sheppard, R. “PFASC – a Parallel Framework for Audio Similarity Clustering”, Proceedings of. TeraGrid '08, Las Vegas, NV. 2008.
17. “MD-GRAPE” In IBM Research, Retrieved January 23, 2012, from http://www.research.ibm.com/grape/
18. Goodale, T., Allen, G., Lanfermann, G., Masso, J., Radke, T., Seidel, E. and Shalf, J. “The Cactus Framework and Toolkit: Design and Applications,” in Vector and Parallel Processing (VECPAR’2002), 5th International Conference, Springer. 2003.

19. "Monte Carlo method" In Wikipedia, Retrieved January 23, 2012, from http://en.wikipedia.org/wiki/ Monte_Carlo_method.
20. Dean, J. and Ghemawat, S. "MapReduce: Simplified Data Processing on Large Clusters," in Proceedings of OSDI '04: 6th Symposium on Operating System Design and Implemention, San Francisco, CA, Dec. 2004.
21. Aspray, W. "John von Neuman and the Origins of Modern Computing", Cambride, Mass.: MIT Press. 1990.
22. Heims, S. J. "The Cybernetics Group", Cambride, Mass.: MIT Press. 1991.
23. Mitchell, M. "Biological Computation", Ubiquity Volume 2011, Article no. 3, 2011.
24. Holland, J. H. "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor. 1975.
25. Kennedy, J.; Eberhart, R. "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks. IV. pp. 1942–1948. 1995.
26. Koza, J. "Genetic Programming: On the programming of computers by means of Natural Selection." MIT Press. 1992.
27. Cristianini, N. and Shawe-Taylor, J. "An Introduction to Support Vector Machines", Cambridge University Press, Cambridge. 2000.
28. Gopu, A., Repasky, R. and McCaulay, S. "Survey of TeraGrid Job Distribution: Toward Specialized Serial Machines as TeraGrid Resources", TeraGrid'07 conference, Madison, WI. 2007.