Building a Parallel Computer System for $18,000 that Performs a Half Peta-Flop per Day

Forrest H Bennett III
Genetic Programming Inc.
Box 1669
Los Altos, California 94023
forrest@evolute.com
http://www.genetic-programming.com

John R. Koza
Section on Medical Informatics
Department of Medicine
Stanford University
Stanford, California 94305
koza@stanford.edu
http://www.smi.stanford.edu/people/koza

James Shipman
IBM Corporation
San Jose, California
JimS@ibm.net

Oscar Stiffelman
Computer Science Department
Stanford University
Stanford, California 94305
ozzie@cs.stanford.edu

ABSTRACT

Techniques of evolutionary computation generally require significant computational resources to solve non-trivial problems of interest. Increases in computing power can be realized either by using a faster computer or by parallelizing the application. Techniques of evolutionary computation are especially amenable to parallelization. This paper describes how to build a 10-node Beowulf-style parallel computer system for $18,000 that delivers about a half peta-flop (10^{15} floating-point operations) per day on runs of genetic programming. Each of the 10 nodes of the system contains a 533 MHz Alpha processor and runs with the Linux operating system. This amount of computational power is sufficient to yield solutions (within a couple of days per problem) to 14 published problems where genetic programming has produced results that are competitive with human-produced results.

1. Introduction

Techniques of evolutionary computation generally require significant computational resources to solve non-trivial problems of interest. Increases in computing power can be realized either by using a faster computer or by parallelizing the application. Computer speeds are expected to continue to double approximately every 18 months in accordance with Moore's law (Moore 1996). Teraflop computers capable of executing 10^{12} floating-point operations per second exist today. Petaflop computers (Sterling, Messina, and Smith 1995; Sterling and Foster 1996a, 1996b; Sterling 1998b; Messina, Sterling, and Smith 1999) capable of executing 10^{15} floating-point operations per second are expected to appear between 2004 and 2007 and to come into general commercial use between 2007 and 2010 (Messina, Sterling, and Smith 1999).

Parallelization provides an opportunity for immediately increasing computing power for applications that can be parallelized efficiently (such as genetic algorithms, genetic programming, and other techniques of evolutionary computation). Amenability to parallelization is a recognized feature of genetic algorithms, genetic programming, and other evolutionary algorithms (Holland 1975; Robertson 1987; Tanese 1989; Goldberg 1989; Stender 1993; Koza and Andre 1995; Andre and Koza 1996a, 1996b).

Section 2 describes one commonly used approach to parallelization of evolutionary algorithms, namely the asynchronous island approach involving semi-isolated subpopulations. Section 3 points out that a half peta-flop of computational power is sufficient to yield a solution (within a couple of days) to over a dozen previously published problems where an evolutionary algorithm has produced results that are competitive with human-produced results. Section 4 describes how to build a 10-node Beowulf-style parallel computer system for $18,000 that delivers about a half peta-flop of computational power per day for runs of genetic programming.

2 Asynchronous Island Approach to Parallelization

In computer runs of evolutionary algorithms, relatively little computer time is expended on tasks such as the creation of the initial population at the beginning of the run and the execution of the genetic operations during the run (e.g., reproduction, crossover, and mutation). The task of measuring the fitness of each individual in each generation of the evolving population is usually the dominant component of the computational burden (with respect to computer time) in solving non-trivial problems of interest using evolutionary algorithms.

These observations give rise to the most commonly used approach to parallelization of evolutionary algorithms, namely the *asynchronous island* model for parallelization. In this approach, the population for a given run is divided into semi-isolated subpopulations (Tanese 1989) called *demes*. Each subpopulation is assigned to a separate processor of the parallel computing system. There are numerous alternative ways to implement this approach. In the usual scenario the run begins with the random creation of the initial population and each individual in a subpopulation is randomly created locally on its local processor. Similarly, the genetic operations are performed locally at each processor. In particular, the selection of individuals to participate in crossover is localized to the processor. The time-consuming task of measuring the fitness of each individual is performed locally at each processor. Upon completion of a generation (or other interval), a relatively small percentage of the individuals in each subpopulation are probabilistically selected (based on fitness) for emigration from each processor to other nearby processors. The processors operate asynchronously in the sense that generations start and end independently at each processor and in the sense that the time of migration is not synchronized. The immigrants to a particular destination typically wait in a buffer at their destination until the destination is ready to assimilate them. The immigrants are then inserted into the subpopulation at the destination processor in lieu of the just-departed emigrants. The overall iterative process then proceeds to the next generation. The guiding principle in implementing this parallel approach is always to fully utilize the computing power of each processor. Thus, for example, if a full complement of immigrants has not yet been received when a processor is ready to assimilate immigrants, the deficiency in immigrants may be made up from randomly chosen copies of the just-departed emigrants. Similarly, if a processor receives two groups of immigrants from a particular other processor before it finishes its current generation, the later immigrants may overwrite the previous immigrants. The inter-processor communication requirements of migration are low because only a modest number of individuals migrate during each generation and because each migration is separated by a comparatively longer periods of time for fitness evaluation. A generation may require from 15 minutes to an hour or more of computer time for many problems involving time-consuming simulations.

Because the time-consuming task of measuring fitness is performed independently for each individual at each processing node, the *asynchronous island* model for parallelization delivers an overall increase in the total amount of work performed that is nearly linear with the number of independent processing nodes. That is, Nearly 100% efficiency is routinely realized when an evolutionary algorithm is run on a parallel computer system using the *asynchronous island* model for parallelization. This near-100% efficiency is in marked contrast to the efficiency achieved in parallelizing the vast majority of computer calculations.

In addition, many researchers have noted that, for many problems, the use of semi-isolated subpopulations with occasional migration often delivers a *super linear* speed-up in terms of the computational effort required to yield a solution (Andre and Koza 1996b). That is, the performance of an evolutionary algorithm is actually enhanced because of the use the island model of parallelization (independent of whether the run is made on a serial or parallel computer).

3 Peta-Cycle Results

There are 14 instances in *Genetic Programming: Darwinian Invention and Problem Solving* (Koza, Bennett, Andre, and Keane 1999; Koza, Bennett, Andre, Keane, and Brave 1999) where genetic programming produced results that are competitive with human-produced results.

When we say that an automatically created result is competitive with one produced by human engineers, designers, mathematicians, or programmers, we mean that it satisfies one or more of the following eight criteria:

(A) The result was patented as an invention in the past, is an improvement over a patented invention, or would qualify today as a patentable new invention.

(B) The result is equal to or better than a result that was accepted and published as a new scientific result at the time when it was published in a peer-reviewed journal.

(C) The result is equal to or better than a result that was placed into a database or archive of results maintained by an internationally recognized panel of scientific experts.

(D) The result is publishable in its own right as a new scientific result (independent of the fact that the result was mechanically created).

(E) The result is equal to or better than the most recent human-created solution to a long-standing problem for which there has been a succession of increasingly better human-created solutions.

(F) The result is equal to or better than a result that was considered an achievement in its field at the time it was first discovered.

(G) The result solves a problem of indisputable difficulty in its field.

(H) The result holds it own or wins a regulated and judged competition involving human contestants (in the form of either live human players or human-written computer programs).

Table 1 tallies the computer time that was consumed by the runs that yielded these 14 instances. For each problem, the table shows the number of minutes needed to create the best-of-run individual for the run and the number of peta-cycles (10^{15} computer cycles) consumed in the run. All problems in table 1 (except for the problem in the last row) were run on a 1995-vintage 64-node Parsytec parallel computer with a 80 MHz PowerPC 601 microprocessor at each processing node. The 64-node Parsytec parallel computer operates at an aggregate rate of 5.12 GHz or 307.2 giga-cycles per minute. The SPECfp95 rating of a PowerPC 601 80-MHz processor is 2.97, so the 64-node Parsytec system delivers about 190 SPECfp95 in the aggregate.

Table 1 Computer time consumed by runs of genetic programming that produced 14 results that are competitive with human-produced results.

	Claimed instance	Minutes	Peta-cycles
1	Transmembrane segment identification problem with architecture-altering operations for subroutines	312	0.096
1	Transmembrane segment identification problem with iteration creation	163	0.050
2	Minimal sorting network (GPPS 1.0)	145	0.045
2	Minimal sorting network (GPPS 2.0)	30	0.009
3	Recognizable Campbell ladder topology for lowpass filters	138	0.042
4	Rediscovery of Zobel's "*M*-derived half section" and "constant K" filter	481	0.148
5	Recognizable Cauer (elliptic) topology for filters	899	0.276
6	Crossover filter	2,673	0.821
6	Crossover filter	5,436	1.670
7	Recognizable voltage gain stage and a Darlington emitter-follower section	1,056	0.324
8	60 dB amplifier	3,139	0.964
8	96 dB amplifier	4,786	1.470
9	Squaring computational circuit	2,504	0.769
9	Cubing computational circuit	2,545	0.782
9	Square root computational circuit	2,817	0.865
9	Cube root computational circuit	2,179	0.669
9	Logarithmic computational circuit	4,309	1.324
9	Gaussian computational circuit (MOSFET)	1,190	0.366
10	Real-time robot controller	22,103	6.790
11	Temperature-sensing circuit	14,204	4.363
12	Voltage reference circuit	37,147	11.412
13	Cellular automata rule for the majority classification problem	4,231	1.300
14	Motifs for the D–E–A-D box family of proteins	3,297	1.013

For reference, the SPECfp95 rating of microprocessors is produced by the Standard Performance Evaluation Corporation (SPEC), a non-profit group of computer vendors, system integrators, universities, and research organizations (`www.specbench.org`). It is designed to provide measures of performance for comparing computationally intensive floating-point workloads on different computer systems. The SPECfp95 rating measures the performance of a computer's processor, memory architecture, and compiler using a suite of existing application and benchmark source code running across multiple platforms. (The last problem in the table was run on a 1994-vintage parallel computer whose speed is about 1/22 of the speed the 64-node Parsytec system).

The 64-node Parsytec machine has a PC Pentium type computer (running Windows 3.11) acting as the host. The host supports the file server, video display, and keyboard for the overall system. Each processing node has an 80-MHz PowerPC 601 microprocessor, 32 megabytes of RAM memory, and an INMOS T805 transputer. There is no disk storage at the processing nodes. The 64 processing nodes do not directly access input-output devices or the host's file system. The 64 nodes run a specialized transputer micro-kernel operating system developed by Parsytec. At each node, the Power PC microprocessor is used for computational purposes, while the transputer is used solely for communication purposes between the processors. The PowerPC microprocessor is based on a RISC architecture that is particularly well-suited for genetic programming work. The 64 processing nodes are arranged in a toroidal network with each processing node communicating with four neighbors. The communication between processing nodes is by means of the one-way, point-to-point channels of the transputer on each processing node. The communication channels are laid out along the physical links between the nodes. The Parsytec virtual router creates a toroidal mesh among the 64 processing nodes. Communication is based on specialized functions contained in the INMOS toolkit.

The 23 runs in table 1 executed an average of 32,797,983 fitness evaluations. These runs averaged 5,034 minutes (about 3.5 days) and consumed about 1.5 petacycles each. This average was heavily influenced by three problems (the temperature-sensing circuit, the voltage reference circuit, and the real-time robot controller).

There are, of course, numerous additional instances where genetic programming and other evolutionary algorithms have been successfully used to evolve programs that are competitive (in the stringent sense used above) with human-produced results. Several such instances have consumed amounts of computer time that are in the neighborhood of a peta-flop For example, Juille's discovery (1995), using evolutionary computation, of a sorting network for 13 items that was smaller than the best network in Knuth (1973) consumed approximately 0.8×10^{14} operations (Juille 1997). Other results (Luke and Spector 1998) consumed substantial (and roughly similar) amounts of computer time. (Of course, there are examples of other instances, such as Juille and Pollack 1998, that consumed lesser amounts of computer time). Our point here is simply that the clustering of over a dozen such human-competitive results, achieved by different researchers, in the general neighborhood of 10^{14} operations suggests that it is reasonable to expect that other such human-competitive results can be attained in the future with this amount of computation. Thus, the ability to build a parallel computer system with capacity of a half peta-flop per day for $18,000 creates an opportunity to produce similar human-competitive results.

4 Building a 10-Node System

This section describes a 10-node parallel computer system with a 533-MHz Alpha microprocessor at each processing node. The 10-node Alpha system operates at about 4.26 GHz in the aggregate (i.e., almost the same as the previously described 1995-vintage 64-node Parsytec parallel computer with a 80 MHz PowerPC 601 microprocessor at each processing node). The SPECfp95 rating of a 533-MHz 21164 Alpha microprocessor is 18.8 (about 6.33 times that of the PowerPC 601 80-MHz microprocessor), so the 10-node Alpha system delivers about 188 SPECft95 (almost identical to the 190 SPECfp95 delivered by 64-node Parsytec system). The Alpha processor has a total of four instruction units. Two of these are integer units and two are floating-point units. The instruction units are pipelined and able to produce a result on every clock cycle if the pipelines are kept full. With the clock running at 533 MHz, the peak performance of the Alpha is $2 \times 533 \times 10^6 = 1.066 \times 10^9$ integer instructions and 1.066×10^9 floating-point instructions per second (i.e., a gigaflop). Peak performance is, of course, usually not realized on a sustained basis in practice on actual code.

Figure 1 shows the various physical elements of a 10-node parallel computer system. The 10-node system is arranged as a computing cluster or Beowulf style system (Sterling 1996, 1998a; Sterling, Salmon, Becker, and Savarese. 1999). The system has a host computer with a 533-MHz Alpha microprocessor with 64 megabytes of RAM (running the Linux operating system). The host contains a 4 GB hard disk, video display, and keyboard. Each of the 10 processing nodes of the system contains a 533-MHz Alpha microprocessor with 64 megabytes of RAM. There is no disk storage at the processing nodes. The processing nodes do not directly access input-output devices or the host's file system. The 10 nodes run the Linux operating system. The 10 processing nodes are arranged in a 2×5 toroidal network with each processing node communicating with four neighbors. The communication between processing nodes is by means of 100 megabit-per-second Ethernet.

Table 2 shows the bill of materials for the 10-node system. Prices that were current as of March 1999 and were provided by Stephen Gaudet of DCG Computers of Londonderry, New Hampshire (`www.dcgomc.com/1999/index.html`). As can be seen from the table, the total price for the entire 10-node system is $18,134 ($1,813 per processing node). The favorable overall cost of the system is based on the fact that all of the items in the bill of materials shown in table 2 are "Commodity Off The Shelf" (COTS) products.

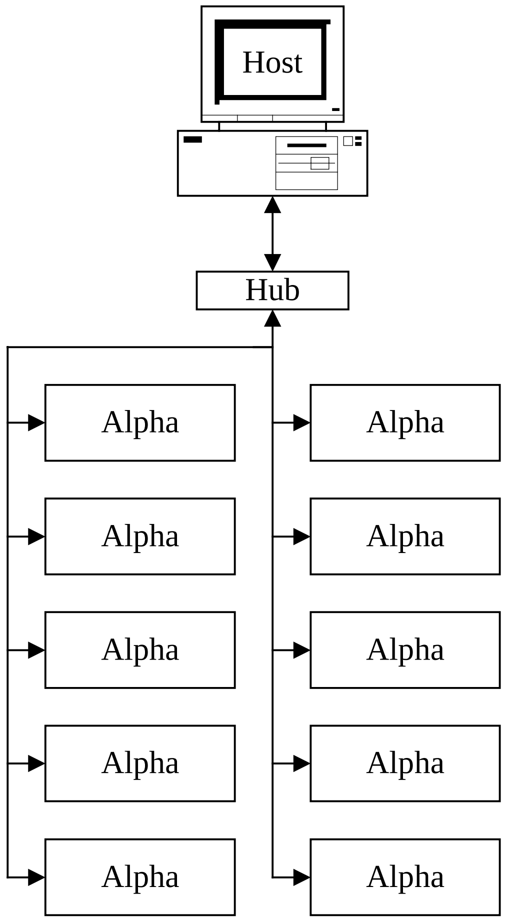

Figure 1 Ten-node parallel computer system.

At the time this was written, the 533 MHz Alpha 164LX processor delivers very favorable floating-point performance in relation to its price. Other Alpha motherboards (such as the RX) will shortly become available which will include an on-board 100 BT Ethernet (thereby eliminating the expense and both of a separate NIC card). There are currently marginally faster Alpha microprocessors in the 21164 series as well as considerably faster Alpha microprocessors in the 21264 series. However, all of these faster chips are (at the time of this writing) priced at a substantial premium (and hence do not deliver the best price-to-performance ratio on a SPECfp95 basis). Favorable pricing is achieved in part by carefully selecting each particular part based on the most recent pricing.

The current advantageousness of the fast Alpha microprocessor is magnified by the role of the fixed-cost items required to support each processing node. These include the case, power supply, network interface card), and the node's share of the cost of the hub.

Approximately half of 64 MB of RAM is available for the storage of the population (with the remainder housing the Linux operating system, the application software, and buffers for exporting and importing individuals, and other items of overhead). Memory is rarely a constraining consideration for the genetic algorithm operating on fixed-length binary character strings; however, it is a consideration for genetic programming. For genetic programming, a population of 32,000 individuals, each occupying 1,000 bytes of RAM can be accommodated with 32 MB of RAM. Using the one-byte-per-point method of storing individual program trees in genetic programming (Andre and Koza 1996a), each individual in the population can possess 1,000 points (functions or terminals). A 10-node system with 64 MB of RAM at each processing node can therefore accommodate a population of 320,000 1,000-point individuals. Depending on the intended size of individuals in the population for the user's particular application, it may be desirable to install more than 64 MB of RAM on each processing node. The cost of the RAM (as shown in table 2) is only about 6% of the total cost of the system and may thus be doubled or quadrupled with relatively little impact on the total cost of the system. The user must carefully consider the likely length of the fitness evaluations that will be performed by the system for his or her particular application in evaluating whether there is any advantage to increasing the amount of RAM per processing node The addition of RAM to house a larger population does not, of course, accelerate the computation at each processing node. If, for example, a fitness evaluation consumes 0.10 seconds (as it may for a simulation), then evaluating the fitness of 32,000 individuals (32,000 1,000-point individuals occupying about 32 MB of RAM) will consume 3,200 seconds. This means that a generation will require 0.89 hours and, consequently, only 27 generations can be executed per day (so that running a mere 100 generations will require four days).

The 100 megabit-per-second Ethernet is more than sufficient to handle the migration of individuals in most practical runs of genetic programming using the island model. Migration usually occurs at a rate of perhaps 1% or 2% in each of four directions on each generation for each processing node. If the population size is 32,000 at each processing node and 2% of the population migrates in each of four directions, then communication of 2,560 individuals (2.56 MB of data if each individual consists of 1,000 bytes) is required for every generation for each processing node. If one generation is processed every 15 minutes (900 seconds), this amounts to transmission of 2,844 bytes (about 23 kilobits) per second for each processing node. This amounts of transmission of 28,440 bytes (about 227 kilobits) per second for 10 nodes. This inter-node communication does not tax a 100 megabit-per-second Ethernet. The Ethernet also easily handles the end-of-generation messages (usually involving less than 10,000 bytes each and occurring only once per generation) from each of the 10 processing nodes to the host processor (as well as other less frequent messages).

The Alpha 164LX processor is available on a motherboard with the ATX form factor. A standard midtower-style case for an Alpha motherboard with the ATX form factor is available as an off-the-shelf commodity product. Such a case solves the electromagnetic emission problems associated with a 533 MHz microprocessor as well as the heat dissipation requirements associated with the Alpha chip. The use of standard cases does not minimize the space occupied by the system; however, it provides a highly cost-effective solution to the emission and heat problems. The standard 230 watt power supplies (produced and priced as a commodity product) are similarly cost-effective. Each processing node has three fans (one for the Alpha microprocessor chip, one for the power supply, and one for the case). The fan on the microprocessor contains a sensor that shuts down the node if it fails.

An Ethernet ("dumb") hub is sufficient for a 10-node system. The price of the one 12-port hub for our 10-node system is $330. In a larger system (such as our 70-node system built in May 1998), Ethernet ("smart") switches are required. We used a Bay Networks BayStack 350T 16-port 10/100 BT Ethernet switch for every 15 processing nodes on our 70-node system. The price of the one 16-port switch for our 70-node system is $2300.

We use an uninterruptable power supply (UPS) providing 15 minutes of support for the system.

Table 2 Bill of materials for 10-node system.

Quantity	Item	Unit price	Total
10	533 MHz Alpha 164LX processor and motherboard	$1,200	$12,000
10	64 MB of SDRAM	$115	$1,150
10	Linux operating systems for nodes	$0	$0
10	100/10 BT DE500-BA Ethernet network interface card (NIC)	$100	$1,000
10	Axxion midtower case with 230 W power supply and fans.	$104	$1,040
1	SMC EZ Hub 12-port Ethernet hub	$330	$330
1	APC Back-UPS Pro 1400 1,400 VA uninterruptable power supply (UPS)	$470	$470
11	Ethernet cables	$4	$44
1	Shelving for 8 nodes	$100	$100
1	Host computer with 64 MB of RAM, 100 BT Ethernet card, 4 GB disk, video display screen, keyboard	$2,000	$2,000
1	Linux operating system for host	$0	$0
		TOTAL	**$18,134**

Linux is, by far, the most common operating system used on individual nodes of Beowulf-style parallel computer systems (whether the nodes are Alpha processors, Pentium processors, or other processors). The Linux operating system is free. In addition, our experience is that the Linux operating system is remarkably robust. Between May 1998 and March 1999, we have operated a 70-node system (designed in the manner described in this paper and composed of 533 MHz Alpha chips) for 11 months on a 24-hour/7-day-per-week basis without a crash (whether due to operating system or electronic equipment). The relatively small size of the Linux operating system obviates the need for disk storage at each processing node. Since the main requirement for memory in genetic programming work is storage of the population and the relatively small genetic programming application, we chose not to have hard disks at each processing node. Diskless booting of the processing nodes is handled by using the BOOTP protocol and configuring the host computer as a BOOTP server.

The host computer receives the end-of-generation reports from each processing node. It creates an output file containing statistics about the run and all pace-setting individuals. This file is stored on the hard disk of the host computer. The host computer described in table 2 is capable of supporting our 70-node system. Since communication between the host processor and the processing nodes is by means of Ethernet, the host computer need not be an Alpha processor and need not employ the Linux operating system.

A 10-node system generates a noticeable amount of heat; however, air conditioning is not usually required for system of this size. The cost of electricity also plays a role in the overall cost of operating the system over its expected useful life.

Additional information about Beowulf-style parallel computer systems can be found in Sterling 1996, Sterling 1998a, and, in particular, in *How to Build a Beowulf: A Guide to Implementation and Application of PC Cluster* (Sterling, Salmon, Becker, and Savarese. 1999). Additional information is also available on the WWW at `http://cesdis.gsfc.nasa.gov/beowulf/consortium.html` and `http://www.cacr.caltech.edu/beowulf/tutorial/building.html`. Electronic mailing lists on Beowulf-style computing include `beowulf-request@cesdis.gsfc.nasa.gov`, `beowulf-announce-request@cesdis.gsfc.nasa.gov`, and `beowulf-wishlist-request@cesdis.gsfc.nasa.gov`.

5 Conclusion

This paper described how to build a 10-node Beowulf-style parallel computer system for $18,000 that delivers about a half peta-flop (10^{15} floating-point operations) of computational power per day for runs of genetic programming or other evolutionary algorithms.

Acknowledgments

Stephen Gaudet of DCG Computers of Londonderry, New Hampshire (`www.dcgomc.com/1999/index.html`) provided extensive helpful information for building the parallel computer system based on Alpha processors. Andy Singleton provided helpful information about the 18-node Beowulf-style parallel system for genetic programming that he built for his own use in New Hampshire in about 1995.

References

Andre, David and Koza, John R. 1995. Parallel genetic programming on a network of transputers. In Rosca, Justinian (editor). *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*. University of Rochester. National Resource Laboratory for the Study of Brain and Behavior. Technical Report 95-2. June 1995. Pages 111–120.

Andre, David and Koza, John R. 1996. Parallel genetic programming: A scalable implementation using the transputer architecture. In Angeline, P. J. and Kinnear, K. E. Jr. (editors). 1996. *Advances in Genetic Programming 2*. Cambridge: MIT Press.

Andre, David and Koza, John R. 1996b. A parallel implementation of genetic programming that achieves super-linear performance. In Arabnia, Hamid R. (editor). *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*. Athens, GA: CSREA. Volume III. Pages 1163-1174.

Goldberg, David E. 1989. Sizing populations for serial and parallel genetic algorithms. In Schaffer, J. D. (editor). *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann Publishers Inc. Pages 70-79.

Holland, John H. 1975. *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press.

Juille, Hugues. 1995. Evolution of non-deterministic incremental algorithms as a new approach for search in state spaces. In Eshelman, L. J. (editor). *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco: Morgan Kaufmann. 351–358.

Juille, Hugues. 1997. Personal communication.

Juille, Hugues and Pollack, Jordan B. 1998. Coevolving the "ideal" trainer: Application to the discovery of cellular automata rules. In Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick L. (editors). *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin*. San Francisco, CA: Morgan Kaufmann. Pages 519 – 527.

Knuth, Donald E. 1973. *The Art of Computer Programming*. Vol. 3. Reading, MA: Addison-Wesley.

Koza, John R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA: MIT Press.

Koza, John R. 1994a. *Genetic Programming II: Automatic Discovery of Reusable Programs.* Cambridge, MA: MIT Press.

Koza, John R. 1994b. *Genetic Programming II Videotape: The Next Generation*. Cambridge, MA: MIT Press.

Koza, John R. 1995. Evolving the architecture of a multi-part program in genetic programming using architecture-altering operations. In McDonnell, John R., Reynolds, Robert G., and Fogel, David B. (editors). *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*. Cambridge, MA: The MIT Press. Pages 695–717.

Koza, John R. and Andre, David. 1995. *Parallel Genetic Programming on a Network of Transputers*. Stanford

University Computer Science Department technical report STAN-CS-TR-95-1542. January 30, 1995.

Koza, John R., Bennett III, Forrest H, Andre, David, and Keane, Martin A. 1999a. *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA: Morgan Kaufmann. Forthcoming.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A., and Brave, Scott. 1999b. *Genetic Programming III Videotape*. San Francisco, CA: Morgan Kaufmann. Forthcoming.

Koza, John R., Bennett III, Forrest H, Andre, David, Keane, Martin A, and Dunlap, Frank. 1997. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*. 1(2). Pages 109 – 128.

Koza, John R., and Rice, James P. 1992. *Genetic Programming: The Movie*. Cambridge, MA: MIT Press.

Luke, Sean and Spector, Lee. 1998. Genetic programming produced competitive soccer softbot teams for RoboCup97. In Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick. (editors). *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin*. San Francisco, CA: Morgan Kaufmann. Pages 214 – 222.

Messina, Paul, Sterling, Thomas, and Smith, Paul H. (editors). 1999. *Petaflops II: Second Conference on Enabling Technologies for Peta(fl)ops Computing, February 15 - 19, 1999, Santa Barbara*.

Moore, Gordon E. 1996. Can Moore's law continue indefinitely? *Computerworld Leadership Series*. 2(6) 2–7. July 15, 1996.

Robertson, George. l987. Parallel implementation of genetic algorithms in a classifier system. In Davis, Lawrence. (editor). *Genetic Algorithms and Simulated Annealing* London: Pittman.

Stender, Joachim (editor). 1993. *Parallel Genetic Algorithms*. Amsterdam: IOS Publishing.

Sterling, Thomas. 1996. The scientific workstation of the future may be a pile of PCs. *Communications of the ACM*. 39(9). September 1996. Pages 11 – 12.

Sterling, Thomas. 1998a. Beowulf-class clustered computing: Harnessing the power of parallelism in a pile of PCs. In Koza, John R., Banzhaf, Wolfgang, Chellapilla, Kumar, Deb, Kalyanmoy, Dorigo, Marco, Fogel, David B., Garzon, Max H., Goldberg, David E., Iba, Hitoshi, and Riolo, Rick L. (editors). *Genetic Programming 1998: Proceedings of the Third Annual Conference, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin*. San Francisco, CA: Morgan Kaufmann. Pages 883 – 887.

Sterling, Thomas 1998b. *Proceedings of Petaflops-Systems Operations Working Review, Bodega Bay, California, June 1 -5, 1998*.

Sterling, Thomas L. and Foster, Ian. 1996a. *Proceedings of Petaflops Architecture Workshop* (PAWS '96), April 22 - 25, 1996.

Sterling, Thomas L. and Foster, Ian. 1996b. .*Proceedings of Petaflops System Software Summer Study (Peta Soft '96), June 17 - 21, 1996*.

Sterling, Thomas L., Salmon, John, and Becker, Donald J., and Savarese, Daniel F. 1999. *How to Build a Beowulf: A Guide to Implementation and Application of PC Clusters*. Cambridge, MA: MIT Press.

Tanese, Reiko. *Distributed Genetic Algorithm for Function Optimization.* PhD. dissertation. Department of Electrical Engineering and Computer Science. University of Michigan. 1989.