PARALLELIZATION

COMPUTER TIME IS THE MOTHER'S MILK OF MACHINE INTELLIGENCE

THE GOOD GNU'S

- **The speed of commercially available single computers continues to double approximately every 18 months in accordance with Moore's law. This exponential growth in computational power (equivalent to two orders of magnitude per decade) is currently expected to continue (and possibly accelerate) over the next decade. Petaflop computing is expected to be available in the next few years.**
- **GA and GP are especially amenable to efficient (and almost effortless) parallelization.**
- **Parallel cluster computer systems can be assembled with relative ease in the Beowulf-style using "Commodity Off-The-Shelf" (COTS) hardware**

THE FITNESS EVALUATION IS THE MOST TIME-CONSUMING PART OF THE VAST MAJORITY OF PROBLEMS OF INTEREST

- **Simulations are typically very time-consuming**
  - **Exception: operations research problems**
- **Simulations often must be made for numerous possible initial conditions**
- **Multiple independent scenarios must be run if the simulation is probabilistic**
- **Time-consuming transition calculations**
- **Time-consuming primitive functions**
- **Large numbers of data points**
- **Many “corners” must be separately tested to account for variations arising from**
  - **temperature**
  - **manufacturing of physical components**
  - **reference voltage**
  - **loads**

THE FITNESS EVALUATION IS THE MOST TIME-CONSUMING PART OF THE VAST MAJORITY OF PROBLEMS OF INTEREST

MORE GOOD GNU'S FOR GA AND GP

- **Fitness evaluation of one individual in the population is not coupled to that of other individuals in the population.**
- **In fact, the evaluation of each separate fitness case is usually not coupled to the evaluation of other fitness cases**

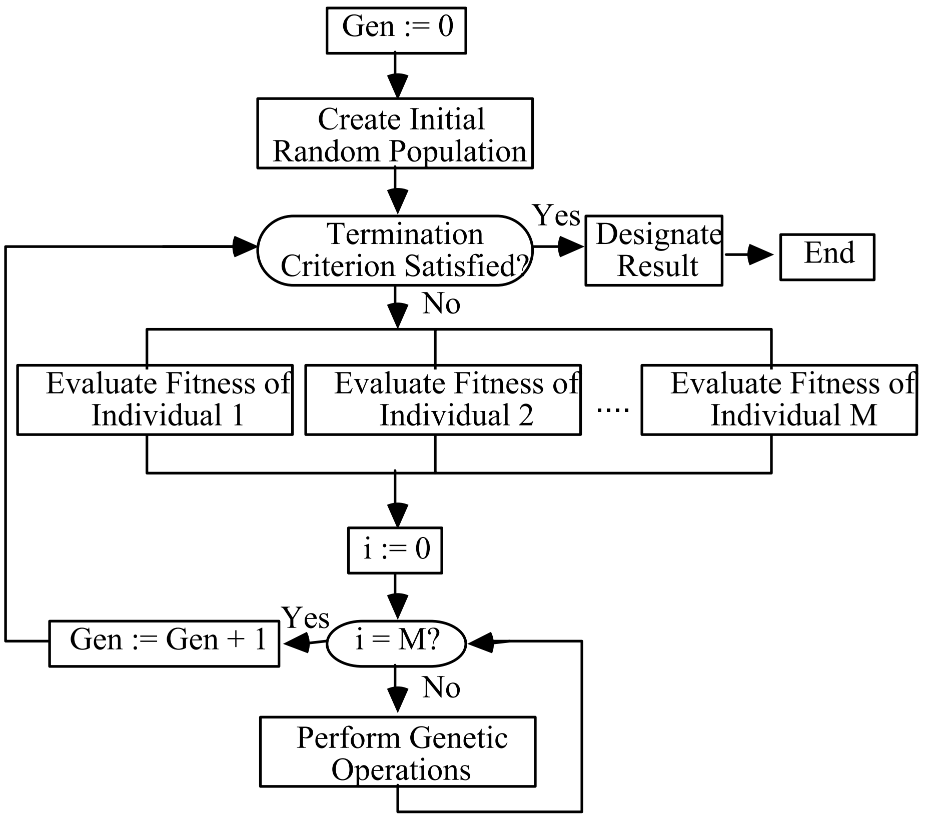
PARALLELIZATION OF GA OR GP

- **Multiple independent runs**
- **Farm-out of individuals**
- **Farm-Out of fitness cases**
- **Semi-isolated subpopulations (also called "demes," "island model of parallelization," or "distributed GA")**

MULTIPLE INDEPENDENT RUNS

- **Adequacy of population size of run**

FARM-OUT OF INDIVIDUALS

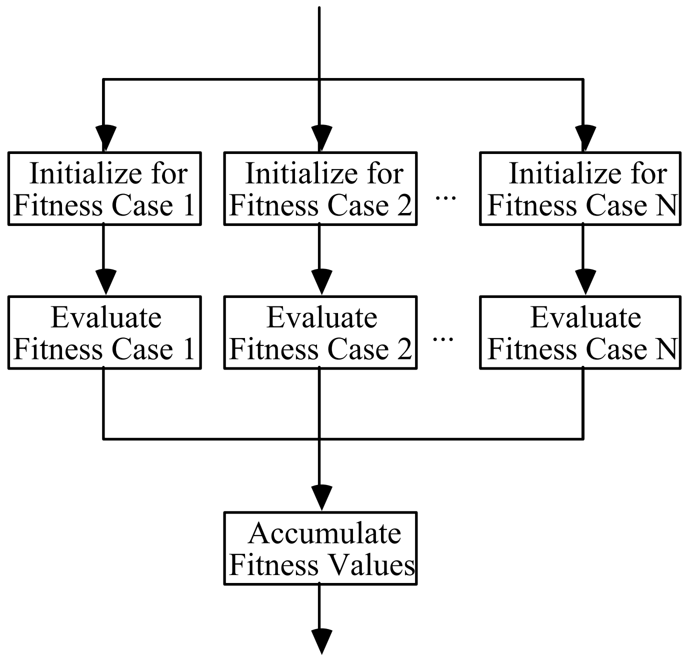

Gen := 0
Create Initial Random Population
Termination Criterion Satisfied?
Yes
Designate Result
End
No
Evaluate Fitness of Individual 1
Evaluate Fitness of Individual 2
....
Evaluate Fitness of Individual M
i := 0
i = M?
Yes
Gen := Gen + 1
No
Perform Genetic Operations

• Works especially well for a moderate number of processors

• Need to match hardware and problem

• Timing issues arising from unequal evaluation times

FARM-OUT OF FITNESS CASES

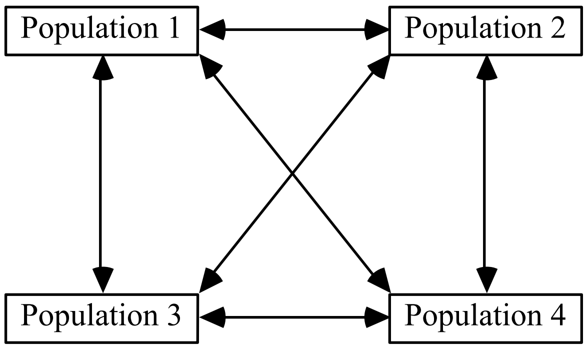

Initialize for Fitness Case 1
Initialize for Fitness Case 2
...
Initialize for Fitness Case N
Evaluate Fitness Case 1
Evaluate Fitness Case 2
...
Evaluate Fitness Case N
Accumulate Fitness Values

• Works especially well for a moderate number of processors

• Need to match hardware and problem

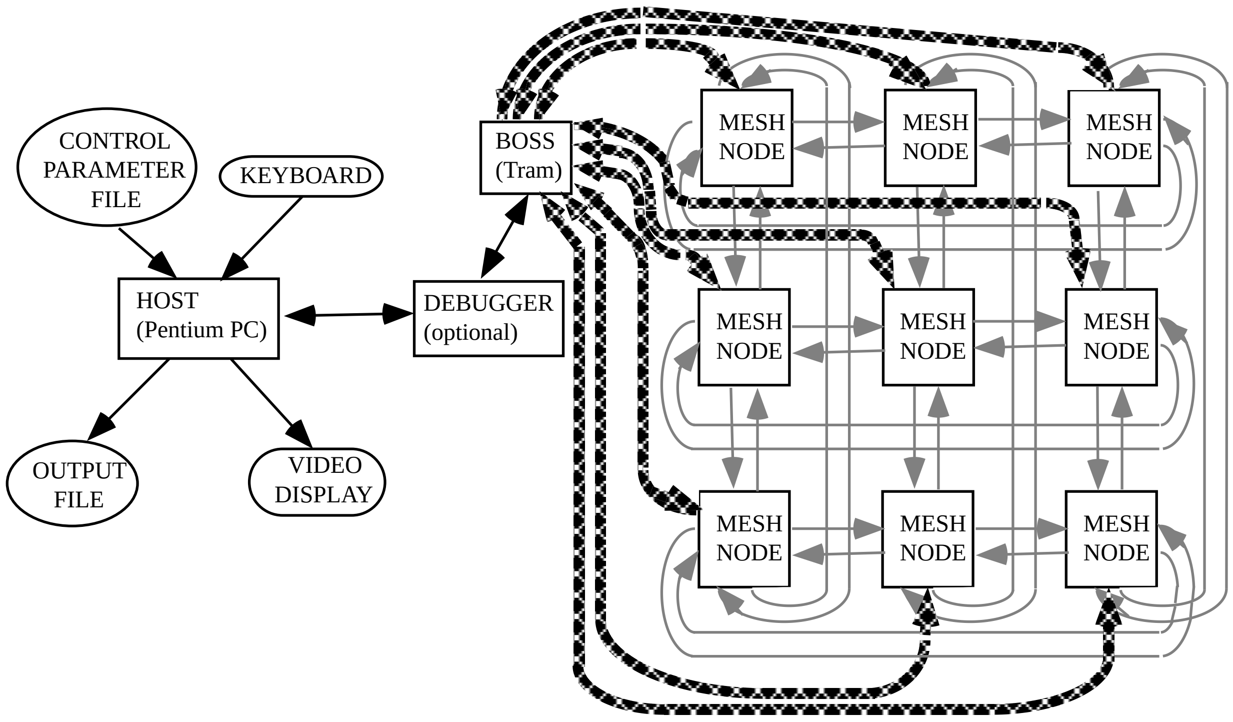
• Timing issues arising from unequal evaluation times

SEMI-ISOLATED SUBPOPULATIONS



- **No synchronization of generations**
- **Occasional small amounts of migration (low bandwidth communication)**
- **Emigrants emigrate (fitness-based selection)**
- **Immigrants arrive, are temporarily buffered, and are assimilated (unfitness-based selection used to make space)**
- **Fault-tolerant**

PARALLELIZATION BY SUBPOPULATIONS ("ISLAND" OR "DEME" MODEL OR "DISTRIBUTED GENETIC ALGORITHM")

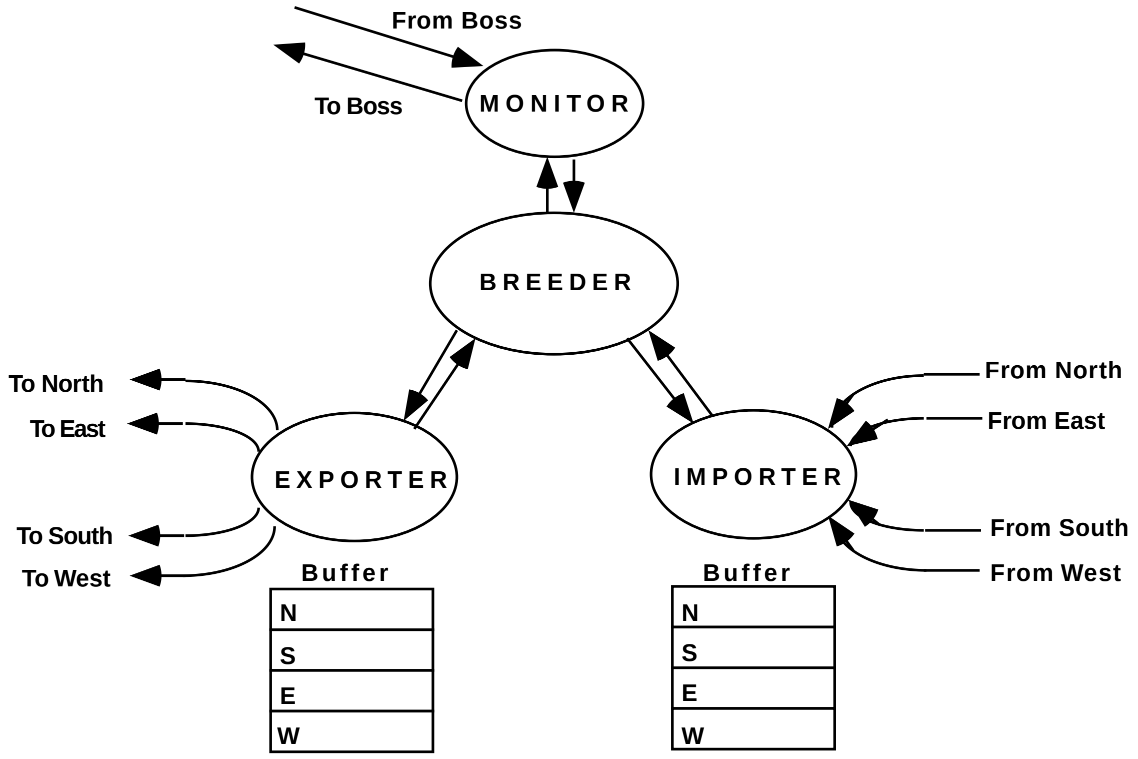


LINES OF COMMUNICATION

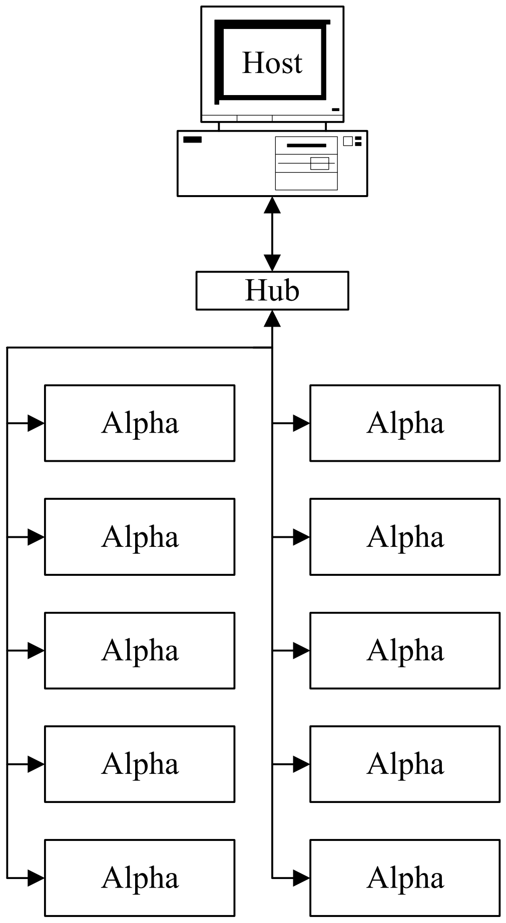
- **between the Boss process and the processing nodes of the network**
- **between (toroidally) adjacent processing nodes of the network**
- **between the Host and the Boss process**

THE FOUR INTER-COMMUNICATING PROCESSES OF EACH OF THE PROCESSING NODES

THE FOUR INTER-COMMUNICATING PROCESSES OF EACH OF THE PROCESSING NODES



10-NODE ALPHA BEOWULF SYSTEM



BILL OF MATERIALS FOR 10-NODE ALPHA BEOWULF SYSTEM (1999)

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**

PRINCIPLES OF PARALLEL GENETIC PROGRAMMING

- **Like Hormel, Get Everything Out of the Pig, Including the Oink**
- **Keep on Trucking**
- **It Takes a Licking and Keeps on Ticking**
- **The Whole is Greater than the Sum of the Parts**

LIKE HORMEL, GET EVERYTHING OUT OF THE PIG, INCLUDING THE OINK

One important guiding principle in implementing parallel genetic programming is to fully utilize the computing power of each processor at all times. Thus, each processor immediately (and separately) begins its main generational loop as soon as it finishes it initial random creation of individuals at the beginning of the run. There is no synchronization of the generations between processors. Similarly, each processor moves on to its next generation, regardless of the status of any other processor. Less than 1% of the processors cycles are consumed by the low-bandwidth migration and communications tasks of the parallel genetic programming algorithm.

KEEP ON TRUCKING

A related principle is that processors should not wait on the achievement of any other event in the system.

- **For example, if immigrants are not received from all (four) of a processor's neighbors at the time when the processor is ready to assimilate immigrants, the processor does not wait for the missing immigrants. It simply assimilates whatever immigrants are in its buffer and moves on. The deficiency in immigrants is made up from randomly chosen copies of the processor's just-departed emigrants.**
- **Similarly, if a processor receives two groups of immigrants from a particular neighboring processor before it is ready to assimilate immigrants, the most recent group of incoming immigrants overwrite the previously received immigrants from the particular neighboring processor (on the principle that later immigrants from a particular subpopulation are, in general, better than earlier immigrants from that same subpopulation).**

IT TAKES A LICKING AND KEEPS ON TICKING

The entire system is designed to be highly fault-tolerant. No processor is considered essential to the run. No acknowledgment is required for any message, including the messages containing emigrants from one processor to another. If a processor fails (e.g., for software or the occasional hardware reason), the boatload of emigrants exported to that processor are simply lost at sea and the run continues.

- **We once rearranged the Ethernet cables between the 1,000 processor during a "live" run. The connecting and disconnecting of wires during the run did not affect the successful completion of the run. Moreover, power cords to many processors were disconnected and reconnected during the run during this rewiring.**

IT TAKES A LICKING AND KEEPS ON TICKING — CONTINUED

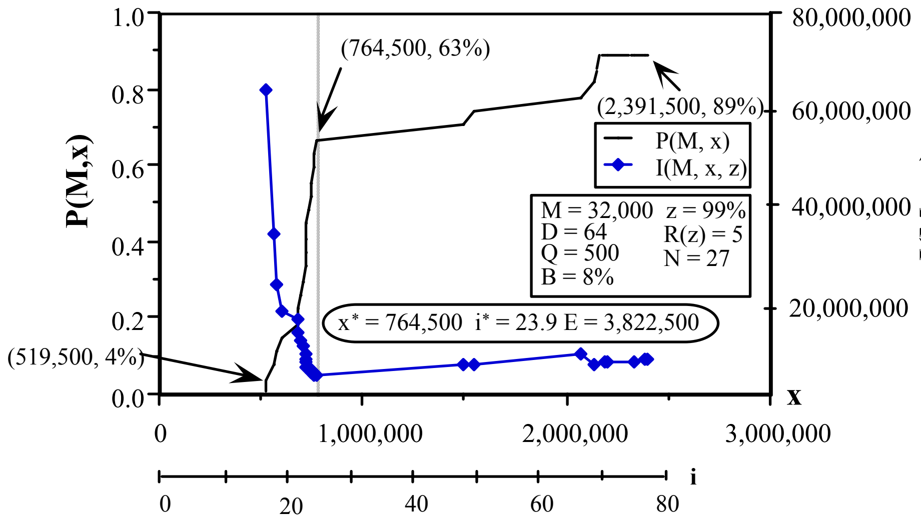
"NOAH" RUNS

By providing a new start-up message to all nodes, the newly reconnected nodes came to life and proceeded to participate in the run. After just one generation, they received emigrants from their neighbors (representing the better individuals of neighboring processors). Darwinian selection immediately favored the newly arrived immigrants (who were, in general, superior to the random individuals on the recently restarted processors) and the genetic operations were consequently performed primarily on the newly arrived immigrants.

THE WHOLE IS GREATER THAN THE SUM OF THE PARTS

Many researchers have noted that, for many problems, genetic programming solves a problem faster with semi-isolated subpopulations and occasional migration than would otherwise be the case. That is, the performance of genetic programming is often enhanced because of the use the island model of parallelization. Thus, it is often said that parallel genetic programming often delivers a *super linear* speed-up in terms of the computational effort required to yield a solution (recognizing that, of course, the benefit of semi-isolated subpopulations can be simulated on a serial computer). In any event, the island model of parallelization is an effective way to make runs of genetic programming.

PERFORMANCE CURVES FOR THE PROBLEM OF SYMBOLIC REGRESSION OF THE BOOLEAN EVEN-5-PARITY FUNCTION WITH *D* = 64 DEMES OF SIZE *Q* = 500 AND A MIGRATION RATE OF *B* = 8%



Approach	Migration rate *B*	Computationa l effort *E*	Best *x**	*P*(*M*,*x*) for best *x**
Panmictic	0%	6,912,000	1,888,000	83%
Parallel	12%	7,072,500	2,357,500	83%
Parallel	8%	3,822,500	**764,500**	63%
Parallel	4%	8,174,000	2,435,999	71%
Parallel	1%	6,594,000	2,198,000	89%

5 PARALLEL COMPUTER SYSTEMS

- **TI LISP machine — 4 nodes (used only for making multiple independent runs)**
- **Transtech — 64 nodes with transputer for both processing and communication**
- **Parsytec — 64 nodes with transputer for communication and 80 MHz PowerPC microprocessor for processing and Parsytec microkernel**
- **Compaq (formerly DEC) — 70 nodes with Alpha 533 MHz processors for processing and 100 BT Ethernet NIC for communication and Linux operating system**
- **1,000-Pentium machine — 1,000 nodes with 350 MHz Pentium II processors for processing and 100 megabit Ethernet NIC for communication and Linux operating system**

5 PARALLEL COMPUTER SYSTEMS

- **Serial Texas Instruments LISP machine**
  - **25 MHz = 2.5 x 10^7 Hz**
  - **2.5 x 10^7 Hz × 86,400 sec./day = 2.16 x 10^{12} cycles/ day**
  - **0.00216 peta-cycles per day (and 463 days = 10^{15} cycles)**
- **64-node Transtech parallel machine**
  - **INMOS T-805 30-MHz transputer ~ Intel 486/33**
  - **Transputer assembly code operates on one-byte**
  - **Effective equivalent for 64 nodes = 2 x 10^{13} cycles / day**
  - 9.25-to-1 speed-up over LISP machine
  - **0.02 peta-cycles per day (and 50 days = 10^{15} cycles)**
- **64-node 80 MHz Parsytec parallel machine**
  - **Power PC 601 processors at 8 x 10^7 Hertz**
  - **8 x 10^7 Hz × 64 nodes = 5.12 x 10^9 Hz**
  - 22-to-1 speed-up over Transtech (based on cycles)
  - **5.12 x 10^9 Hz/sec. × 86,400 sec./day =4.4 x 10^{14} Hz/ day**
  - **0.44 peta-cycles per day (and 2.3 days = 10^{15} cycles)**

5 PARALLEL COMPUTER SYSTEMS

- **70-node 533 MHz Alpha parallel machine**
  - **5.33×10^8 Hz $\times$ 70 nodes = 3.73×10^{10} Hz = 0.03 tera-Hz**
  - 7.2-to-1 speed-up over Parsytec (based on cycles)
  - **3.73×10^{10} Hz $\times$ 86,400 = 3.2×10^{15} cycles/ day**
  - **So, 3.2 peta-cycles per day**
  - **NOTE: The Alpha processor typically yields about 3 instructions per cycle on a GP application, so this system operates at about 0.09 tera-ops**

- **1,000-node 350 MHz Pentium II machine**
  - **3.5×10^8 Hz $\times$ 1,000 = 3.5×10^{11} Hz**
  - **3.5×10^{11} Hz $\times$ 86,400 sec./day = 3.02×10^{16} cycles/ day**
  - 9.4-to-1 speed-up over Alpha machine (based on cycles)
  - 13,633-to-1 speed-up over LISP (based on cycles)
  - **So, 30 peta-cycles per day**
  - **NOTE: The Pentium II processor typically yields about 3 instructions per cycle on a GP application, so this system operates at about 1.05 tera-ops**

PETA-OPS

- **Human brain operates at 10^{12} neurons operating at 10^{3} per second = 10^{15} ops per second**
- **10^{15} ops = 1 peta-op = 1 bs (brain second)**

PETAFLOPS COMPUTING INITIATIVE

- **10^{15} operations PER SECOND by 2007**
- **Commercial petaflops by 2010**

COMPUTER TIME FOR RUNS PRODUCING HUMAN-COMPETITIVE RESULTS FROM GP-3 BOOK (1999)

Claimed instance	Population *M*	Generation *i*	*M*(i+1)*	Minutes	Peta-cycles
Transmembrane segment identification problem with architecture-altering operations for automatically defined functions	**128,000**	**28**	**3,712,000**	**312**	**0.096**
Transmembrane segment identification problem with iteration creation	**64,000**	**42**	**2,752,000**	**163**	**0.050**
Minimal sorting network (GPPS 1.0)	**640,000**	**31**	**20,480,000**	**145**	**0.045**
Minimal sorting network (GPPS 2.0)	**300,000**	**33**	**10,200,000**	**30**	**0.009**
Recognizable ladder topology of Butterworth or Chebychev lowpass and highpass filters	**320,000**	**49**	**16,000,000**	**138**	**0.042**
Recognizable "bridged T" topology for filters	**320,000**	**53**	**17,280,000**	**481**	**0.148**
Recognizable elliptic topology for filters	**640,000**	**31**	**20,480,000**	**899**	**0.276**
Crossover filter	**640,000**	**137**	**88,320,000**	**2,673**	**0.821**
Crossover filter	**640,000**	**158**	**101,760,000**	**5,436**	**1.670**
Recognizable voltage gain stage and a Darlington emitter-follower section	**640,000**	**45**	**29,440,000**	**1,056**	**0.324**
60 dB amplifier	**640,000**	**109**	**70,400,000**	**3,139**	**0.964**
96 dB amplifier	**640,000**	**86**	**55,680,000**	**4,786**	**1.470**
Squaring computational circuit	**640,000**	**37**	**24,320,000**	**2,504**	**0.769**
Cubing computational circuit	**640,000**	**74**	**48,000,000**	**2,545**	**0.782**
Square root computational circuit	**640,000**	**71**	**46,080,000**	**2,817**	**0.865**
Cube root computational circuit	**640,000**	**60**	**39,040,000**	**2,179**	**0.669**
Logarithmic computational circuit	**640,000**	**55**	**35,840,000**	**4,309**	**1.324**
Gaussian computational circuit (MOSFET)	**640,000**	**36**	**23,680,000**	**1,190**	**0.366**
Real-time robot controller	**640,000**	**31**	**20,480,000**	**22,103**	**6.790**
Temperature-sensing circuit	**640,000**	**25**	**16,640,000**	**14,204**	**4.363**
Voltage reference circuit	**640,000**	**80**	**51,840,000**	**37,147**	**11.412**
Cellular automata rule for the majority classification problem	**51,200**	**17**	**921,600**	**4,231**	**1.300**
Motifs for the D–E–A-D box family of proteins	**256,000**	**42**	**11,008,000**	**3,297**	**1.013**

COMPUTER TIME FOR 41 RUNS IN GP-4 BOOK (2003)

Problem	Total population	Generation *i*	*M**(*i* +1)	Total hours	Nodes	Hertz	Petacycles
Two-lag plant	66,000	32	2,178,000	44.5	66	3.5×10^{10}	5.6
Three-lag plant	66,000	31	2,112,000	32	66	3.5×10^{10}	4.0
Three-lag plant with five second delay	500,000	126	63,500,000	65	1,000	3.5×10^{11}	81.9
Non-minimal phase plant	66,000	38	2,574,000	46	66	3.5×10^{10}	5.8
RC circuit with gain greater than two	660,000	927	612,480,000	24	66	3.5×10^{10}	3.0
Philbrick circuit	660,000	39	26,400,000	7	66	3.5×10^{10}	0.9
NAND circuit	132,000	17	2,376,000	10	66	3.5×10^{10}	1.3
Arithmetic logic unit (ALU) circuit	1,320,000	33	44,880,000	170	66	3.5×10^{10}	21.4
Square root computation al circuit	10,000,000	66	670,000,000	52	1,000	3.5×10^{11}	65.5
Lowpass filter without an explicit test fixture	1,000,000	211	212,000,000	9	1,000	3.5×10^{11}	11.3
Lowpass filter with layout	1,120,000	138	155,680,000	28	56	2.9×10^{10}	3.0
Amplifier with layout	10,000,000	101	1,020,000,0 00	17	1,000	3.5×10^{11}	21.4
Yagi-Uda antenna	500,000	90	45,500,000	22	1,000	3.5×10^{11}	27.7
Metabolic pathway for phospholipid cycle	100,000	225	22,600,000	17	1,000	3.5×10^{11}	21.4
Metabolic pathway for ketone bodies	100,000	97	9,800,000	20	1,000	3.5×10^{11}	25.2
Three-lag plant with free variable	500,000	42	21,500,000	23.4	1,000	3.5×10^{11}	29.5
Controller for two	100,000	217	21,800,000	40.6	1,000	3.5×10^{11}	51.2

families of plants							
Zobel network problem (two free variables)	**50,000,000**	**16**	**8,500,000**	**2**	**1,000**	**3.5×10^{11}**	**3.0**
Third-order elliptic lowpass filter with a free variable for the modular angle	**1,000,000**	**293**	**294,000,000**	**71**	**1,000**	**3.5×10^{11}**	**89.5**
Passive lowpass filter with s free variable for the passband boundary	**10,000,000**	**78**	**790,000,000**	**45**	**1,000**	**3.5×10^{11}**	**56.6**
Active lowpass filter with variable passband boundary with free variable	**5,000,000**	**101**	**501,000,000**	**151**	**1,000**	**3.5×10^{11}**	**190.3**
Lowpass/highpass filter with free variables	**10,000,000**	**47**	**480,000,000**	**45**	**1,000**	**3.5×10^{11}**	**56.6**
Lowpass/highpass filter with variable passband boundary with a free variable	**10,000,000**	**93**	**940,000,000**	**84**	**1,000**	**3.5×10^{11}**	**105.8**
Variable quadratic/cubic computational circuit (one free variable)	**1,000,000**	**241**	**242,000,000**	**49**	**1,000**	**3.5×10^{11}**	**61.7**
Variable 40–60 dB amplifier (one free variable)	**2,000,000**	**332**	**666,000,000**	**50**	**1,000**	**3.5×10^{11}**	**63.0**
Improved PID tuning rules (four free variables)	**100,000**	**76**	**7,700,000**	**107**	**1,000**	**3.5×10^{11}**	**134.6**

First non-PID parameterized controller (two free variables)	**100,000**	**88**	**8,900,000**	**320**	**1,000**	**3.5×10^{11}**	**403.2**
Second non-PID parameterized controller (two free variables)	**100,000**	**38**	**3,900,000**	**397**	**1,000**	**3.5×10^{11}**	**500.2**
Third non-PID parameterized controller (two free variables)	**100,000**	**199**	**20,000,000**	**692**	**1,000**	**3.5×10^{11}**	**871.9**
Reinvention of negative feedback	**1,000,000**	**48**	**49,000,000**	**7**	**1,000**	**3.5×10^{11}**	**8.8**
Low-voltage balun circuit	**5,000,000**	**97**	**490,000,000**	**25**	**1,000**	**3.5×10^{11}**	**31.5**
Mixed analog-digital variable capacitor circuit	**2,000,000**	**98**	**198,000,000**	**88**	**1,000**	**3.5×10^{11}**	**110.9**
High-current load circuit–First run	**2,000,000**	**114**	**230,000,000**	**134**	**1,000**	**3.5×10^{11}**	**168.8**
High-current load circuit–Second run	**2,000,000**	**215**	**432,000,000**	**67**	**1,000**	**3.5×10^{11}**	**84.4**
Voltage-current conversion circuit	**5,000,000**	**109**	**550,000,000**	**83**	**1,000**	**3.5×10^{11}**	**104.6**
Cubic function generator–First run	**5,000,000**	**182**	**915,000,000**	**206**	**1,000**	**3.5×10^{11}**	**259.6**
Cubic function generator–Second run	**2,000,000**	**326**	**654,000,000**	**135**	**1,000**	**3.5×10^{11}**	**170.1**
Tunable integrated active filter–First run	**2,000,000**	**70**	**142,000,000**	**23**	**1,000**	**3.5×10^{11}**	**29.0**
Tunable integrated active filter–Second run	**2,000,000**	**50**	**102,000,000**	**14**	**1,000**	**3.5×10^{11}**	**17.6**

Tunable integrated active filter– Third run	**2,000,000**	**38**	**78,000,000**	**12**	**1,000**	**3.5×10^{11}**	**15.1**
Tunable integrated active filter– Fourth run	**2,000,000**	**27**	**56,000,000**	**6**	**1,000**	**3.5×10^{11}**	**7.6**
Average	**3,530,000**	**128.7**	**267,000,000**	**81.9**			**93.4**

DISCUSSION — COMPUTER TIME FOR 41 RUNS IN GP-4 BOOK (2003)

- **The 41 runs ran for an average elapsed time of 81.9 hours (3.4 days) and consumed an average of 93.4 petacycles**

- **The number of petacycles consumed by the runs range over about three orders of magnitude—from the 3.0 petacycles consumed by the relatively simple Zobel network problem to the 871.9 petacycles (almost 10^{18} cycles) consumed by the four-week run that produced the parsimonious (third) non-PID controller.**

GP-4 BOOK DISCUSSION — CONTINUED

• If available computer speed continues to double approximately every 18 months in accordance with Moore's law for the next decade, a computation running for an elapsed time of 81.9 hours (the average from table 17.1) today will require only about 49 minutes (about 1% as much elapsed time) a decade from now.

• A computation similar to the most time-consuming run (the four-week run that produced the third non-PID parameterized controller) will require only about 7 hours of elapsed time a decade from now.

• The runs involving the six post-2000 patented inventions ran for an average elapsed time of 80 hours each (very close to the overall average of 81.9 hours).

• If Moore's law is applied to these runs, a similar run would require only about 48 minutes of elapsed time a decade from now.

COMPARISON BETWEEN RUNS IN GP-3 AND GP-4 BOOKS

- **Comparing the 41 runs from *Genetic Programming IV* book (2003) with the 14 runs involving human-competitive results shown in *Genetic Programming III* book, the elapsed time for the two groups of runs was about the same (3.4 days for GP-4 versus 3.5 days for GP-3 book).**

COMPARISON BETWEEN RUNS IN GP-3 AND GP-4 BOOKS — CONTINUED

- **However, the problems treated in GP-4 book were qualitatively more substantial than those in GP-3 book.**
  - **The average population size for the 41 runs in GP-4 is 3,530,000 whereas it was only 256,000 in GP-3.**
  - **The 41 runs in GP-4 entailed about 8 times as many fitness evaluations as those in GP-3 (257,000,000 for GP-4 versus 32,800,000 for GP-3 book).**
  - **Moreover, the 41 runs in GP-4 consumed an average of about 62 times more computational resources than those in GP-3 (93.4 petacycles for GP-4 versus 1.5 petacycles for GP-3 book).**

COMPARISON BETWEEN RUNS IN GP-3 AND GP-4 BOOKS — CONTINUED

- **The fitness measures for problems in GP-4 are generally far more time-consuming than those in GP-3 because the fitness measures in GP-4 contain a considerably greater number of different elements (up to a high of 193 elements for the third non-PID controller in section 13.2.3) and because the elements of the fitness measures in GP-4 usually involve time-domain simulations (which generally consume far more computer time than the frequency-domain simulations and other relatively fast types of simulation and evaluation found in GP-3 .**

FOOTNOTES TO DATA ABOUT 41 RUNS IN GP-4 BOOK

• No adjustment was made in the durational data for the fact that, in practice, no fitness evaluations are actually performed for the percentage (usually about 9%) of the population that is reproduced in each generation or for the (smaller) percentage of the offspring produced by semantics-preserving architecture-altering operations (e.g., subroutine duplication or branch creation).

• The generation number during which the best-of-run individual was first created on its processing node. However, because of the asynchronous operation of the processing nodes in the parallel system, other processing nodes are, at that same moment, generally at a somewhat earlier or somewhat later generation number.

FOOTNOTES TO DATA ABOUT 41 RUNS IN GP-4 BOOK — CONTINUED

- **It is not uncommon for a small fraction of the 1,000 processing nodes to stop during a particular run because of a deeply-buried problem-specific software flaw that becomes exposed by the execution of the run's hundreds of millions of genetic operations.**
- **Tt is not uncommon for some nodes to be out of commission at any given time because of fan failures (the only hardware problem that we regularly encounter).**
- **No adjustment was made to reflect the fact that we frequently commandeer 10 nodes of the 1,000-node system in the middle of a run in order to test a future run's software.**

FOOTNOTES TO DATA ABOUT 41 RUNS IN GP-4 BOOK — CONTINUED

- **In the last three cases, all individuals on the adversely affected processing nodes as well as all individuals that attempt to emigrate to the affected nodes are simply lost.**
- **None of these five factors are material to the statistics concerning the overall operation of a 1,000-node system.**

COMMERCIAL PRACTICALITY OF GENETIC PROGRAMMING FOR AUTOMATED CIRCUIT SYNTHESIS BASED ON THE 6 POST-2000 PATENTED CIRCUITS IN GP-4 BOOK (2003)

Problem	**Population *M***	**Generation *i***	***M**(*i* +1)**	**Total hours**
Low-voltage balun circuit	**5,000,000**	**97**	**490,000,000**	**25**
Mixed analog-digital variable capacitor circuit	**2,000,000**	**98**	**198,000,000**	**88**
High-current load circuit–First run	**2,000,000**	**114**	**230,000,000**	**134**
High-current load circuit–Second run	**2,000,000**	**215**	**432,000,000**	**67**
Voltage-current conversion circuit	**5,000,000**	**109**	**550,000,000**	**83**
Cubic function generator–First run	**5,000,000**	**182**	**915,000,000**	**206**
Cubic function generator–Second run	**2,000,000**	**326**	**654,000,000**	**135**
Tunable integrated active filter–First run	**2,000,000**	**70**	**142,000,000**	**23**
Tunable integrated active filter–Second run	**2,000,000**	**50**	**102,000,000**	**14**
Tunable integrated active filter–Third run	**2,000,000**	**38**	**78,000,000**	**12**
Tunable integrated active filter–Fourth run	**2,000,000**	**27**	**56,000,000**	**6**

COMMERCIAL PRACTICALITY OF GP FOR POST-2000 PATENTED CIRCUITS — CONTINUED

- **The average number of hours for runs involving the six post-2000 patented circuits is 25, 88, 99, 83, 170, and 14, respectively. The average of these averages is 80 hours (3.3 days). We use the average of the averages here because we made four runs of the problem that took the least computer time.**
- **All 6 problems were run on a home-built parallel computer system consisting of 1,000 350-MHz Pentium II processors. This system operates at an overall rate of 3.5×10^{11} Hz. A 3.3-day run represents about 10^{17} cycles (i.e., 100 petacycles).**

COMMERCIAL PRACTICALITY OF GP FOR POST-2000 PATENTED CIRCUITS — CONTINUED

- **The relentless iteration of Moore's law promises increased availability of computational resources in future years. If available computer capacity continues to double approximately every 18 months over the next decade, a computation requiring 80 hours will require only about 1% as much computer time (i.e., about 48 minutes) a decade from now.**

COMMERCIAL PRACTICALITY OF GP FOR POST-2000 PATENTED CIRCUITS — CONTINUED

- **Aside from the promise of increased availability of computational resources in the future, there are two reasons why we are currently not near the boundary of the current capability of genetic programming to automatically synthesize analog circuits.**
    - **Multiple runs of a probabilistic algorithm are often necessary to solve a problem. However, ignoring partial runs used for debugging purposes, <u>all 11 runs</u> involving the six post-2000 patented circuits produced a satisfactory solution. The success rate of 100% is thus unusual with a probabilistic algorithm. This success rate suggests that we are currently not near the boundary of the current capability of genetic programming.**

COMMERCIAL PRACTICALITY OF GP FOR POST-2000 PATENTED CIRCUITS — CONTINUED

- **The (largely intentional) inefficiency of the runs is a second indication. As previously mentioned, the runs of genetic programming in GP-4 book were undertaken with two distinct (and conflicting) orientations. One orientation involves using as little human-supplied knowledge, information, analysis, and intelligence as possible. The other is the engineer's orientation to solve problems as quickly and efficiently as possible.**

This dominant orientation is irrelevant to the practicing engineer.

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

- **First, each of the problems could have been solved much more efficiently by using a more customized initial circuit. Our use of the floating embryo on the six post-2000 patented circuits enabled us to minimize the differences in the human-supplied preparatory steps among the six problems. This uniformity helps to persuade the reader that the human user need employ only *de minimus* information and domain knowledge in order to launch a run of genetic programming. However, the floating embryo used on all six problems in this chapter is manifestly inefficient.**

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

- **Second, the components that are inserted into a developing circuit need not be as primitive as a single transistor, resistor, or capacitor. Instead, the set of component-creating functions could easily be expanded to include numerous frequently-used combinations of components. Potentially useful combinations of components include voltage gain stages, Darlington emitter-follower sections, current mirrors, cascodes, three-ported voltage divider subcircuits composed of two resistors in series, and three-ported subcircuits consisting of two resistors (or capacitors) with their common point connected to power or ground. For certain problems, the set of primitives could readily be expanded to include higher-level entities, such as filters, op amps, oscillators, voltage-controller current sources, multipliers, and phase-locked loops.**

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

• Third, although the runs of the six post-2000 patented circuits were intentionally done in a uniform way, a practicing engineer has no reason to enforce such uniformity. For example, we did not use automatically defined functions for any of the six problems. However, most practical electrical circuits are replete with reuse. A practicing engineer would recognize that reuse is specifically important in at least two of the six problems (namely the mixed analog-digital integrated circuit for variable capacitance and the low-voltage high-current transistor circuit for testing a voltage source). The practicing engineer would have no reason to forgo the manifest benefits of reuse and automatically defined functions.

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

• Fourth, considerable work has been done in recent years to accelerate the convergence characteristics of circuit simulators. As a result, there are numerous commercially available simulators that are considerably more efficient than the we used. We use a version of the SPICE3 simulator (Quarles, Newton, Pederson, and Sangiovanni-Vincentelli 1994) that we modified in various ways. Speedups of up to 10-to-1 are reportedly possible today.

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

• Fifth, there are numerous opportunities to incorporate problem-specific knowledge into a run of genetic programming. For example, the developmental process need not start merely with modifiable wires. A substructure of known utility for a particular problem can be hard-wired into the embryo, thereby relieving genetic programming of the need to reinvent it.

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

- **Sixth, it is possible to integrate general knowledge of electrical engineering into a run of genetic programming. For example, Sripramong and Toumazou (2002) combine current-flow analysis into their runs of genetic programming for the purpose of automatically synthesizing CMOS amplifiers.**

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

- **Seventh, the efficiency of runs can be improved by adapting several of the principles set forth in *The Design of Innovation: Lessons from and for Competent Genetic Algorithms* (Goldberg 2002) to the domain of genetic programming.**

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

- **Eighth, because there usually are multiple competing elements in the fitness measures of problems of circuit synthesis in the real world, the efficiency of runs of genetic programming can be improved by using some of the recently published new techniques of multiobjective optimization (Osyczka 1984; Bagchi 1999; Deb 2001; Coello Coello, Van Veldhuizen, and Lamont 2002; and Zitzler, Deb, Thiele, Coello Coello, and Corne 2001).**

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

- **Ninth, the efficiency of runs can be improved by adapting some of the innovative ideas in *Efficient and Accurate Parallel Genetic Algorithms* (Cantu-Paz 2000) to the domain of genetic programming.**

ACCELERATION OF RUNS OF POST-2000 PATENTED CIRCUITS

• Tenth, there has been an outpouring of theoretical work in the past few years on the theory of genetic algorithms and genetic programming. Many of the insights in *Foundations of Genetic Programming* (Langdon and Poli 2002) can be used to improve the efficiency of runs of genetic programming.