Lecture 20: Generic Algorithms

CS178: Programming Parallel and Distributed Systems

April 16, 2001
Steven P. Reiss

I. Overview

II. Genetic Algorithms

A. Basic Concepts

1. Nature does a good job of finding near-optimal solutions to problems through evolution

a) Using lots of time

2. It does this through a variety of mechanisms

a) First selecting parents (survival of the fittest)

b) Then by merging genes from the parents (crossover)

c) Finally by introducing random mutations

3. Generic approach to search tries to mimic this

a) Need to define analogs to fitness, crossover, mutation

b) But then just simulate lots of generations

4. This is useful here because it is easy to parallelize

B. Constraints

1. You want to have some movement toward a solution

a) Through selecting parents

2. But not too much

a) Otherwise you'll get to local maximum rather than global

3. You want to ensure stability

a) If you have two near-optimal solutions, offspring will be as well

4. Mutations should be at all levels

a) Major -- to force jumping around in the solution space

b) Intermediate -- jumping around in solution space

c) Minor -- to handle local hill climbing

5. **You need to go through many generations**

C. Algorithm

```
generation = 0
setup initial Population(generation)
evaluate Population(generation)
while (not terminationCheck()) {
   ++generation;
   select Parents(generation) from Population(generation-1)
   apply crossover to Parents(generation) to get
         Offspring(generation)
   apply mutation to Offspring(generation) to get
         Population(generation)
   evaluate Population(generation)
end
```

1. **Representation**
   a) For this to work you need to create a compact representation of a solution
      (1) Typically a string of bits
   b) Must allow the above operations
      (1) Random generation of an initial solution
      (2) Ability to do consistent crossover of solutions
      (3) Ability to apply mutations in consistent ways
   c) Must keep legal in some way
2. **Example -- maximize polynomial in x,y,z**
   a) We have three numbers, x,y,z to be represented
   b) Crossover methods -- single, multiple splits
   c) Mutation -- change random bit
3. **Example -- how might you to TSP**
   a) Numbers giving priority
   b) Numbers giving next (done with checking for used, find next using rehash methods)
4. **Choosing parents**
   a) Want some bias on fitness, but not too much
   b) k-tournaments (choose k at random, pick best)
5. **Termination conditions**
   a) When best changes little
   b) After a fixed number of generations

6. **Variations**
    a) Keep the best parents in the next generation
    b) Population size can be fixed or vary
    c) Mutation rate

D. Parallel algorithm

1. **Much of this can be parallelized**
    a) But not necessarily directly
    b) Don't want to distribute everything each generation
    c) But then nature doesn't either
2. **Work with islands (separated populations) that occassionally intermingle**
    a) Add a migration component to the loop
    b) Migrate best versus random
        (1) Best might put too much pressure on
        (2) Best every k times, random otherwise
    c) This can be applied each time, every k times, or random
3. **Migration to where**
    a) Could broadcast the best items to all other nodes
        (1) Island model
    b) Alternatively, just pass it on to local nodes
        (1) Stepping-stone model
4. **Termination is more difficult**

E. Overall

1. **Genetic algorithms can work quite effectively**
    a) Finding the right representation can be difficult
    b) They require a lot of tuning for the individual problem
2. **Relatively easy to parallelize**
    a) Very effective parallelization
    b) Little communication overhead
3. **Not necessarily intuitive**

III. Successive Refinement

A. Basic concept

1. **Look at the search space in a coarse way**
   a) Bit-wise, etc.
2. **Do a full search here**
3. **Choose the best solution(s), and do a search**
4. **Essentially the same as above, but with intelligent choice of starting points**

B. Again a problem of representation

1. **Need to represent the source in a step-wise fashion**
2. **For 3 numbers problem**
   a) Bitwise or scalled representation
3. **For traveling salesman problem**
   a) Break into clusters
   b) Treat each cluster as a node (with appropriate weights
   c) This can give starting points for local search

C. Parallelization

1. **Initial grid can be handed out to different processors**
2. **Find best K solutions after these are done**
3. **Then these are redistributed for more refined search**
4. **The process is repeated**

IV. Performance Analysis

A. Parallel programming is about performance

1. **The idea is to solve large problems**
2. **Efficiency is a primary concern**

B. Components of parallel performance

1. **Execution time**
   a) Serial performance
   b) Algorithms and data structures
2. **Input/Output time**
3. **Communications overhead**
   a) Latency -- fixed cost to establish communication

b) Bandwidth -- cost per data unit

C. Hardware differences

1. These tend to vary and change a lot

a) Vary between architectures

b) Vary within an architecture

c) Change over time

2. Parameters

a) Ta -- cost of an arithmetic operation

b) Ts -- cost of message setup

c) Tc -- cost of sending 8 bytes

3. Ethernet/Suns

a) Ta = 0.0025

b) Ts = 500

c) Tc = 10

4. IBM SP/2

a) Ta = 0.0042

b) Ts = 35

c) Tc = 0.23

5. Designing a program in general is difficult

a) Most often tuned to a particular architecture

D. Software performance analysis

1. Single process analysis

a) Can be used for identifying computational bottlenecks

b) Using standard tools like prof/gprof

2. Getting timing information

a) double MPI_Wtime()

(1) Returns number of seconds from arb point in past

(2) This is real time

b) double MPI_Wtick()

(1) Returns the precision of MPI_Wtime()

c) Add calls to MPI_Barrier(MPI_Comm)

(1) To see where processors are at a point

(2) To surround the region being timed

E. Tools

1. Most MPI implementations come with tools

a) Help in understanding how the system is behaving

b) What the different processes are doing when

2. MPI has builtin hooks for doing tracing for this purpose

3. For the Suns: run /cs/src/mpi/lam/bin/xmpi

a) This lets you start the application up

b) Hitting trace will produce a trace diagram

c) You can explore the other options

F. Performance techniques

1. Pipeline communication as much as possible

a) Avoiding collisions

2. Overlap communication with computation

3. Keep the load balanced