CSE/ECE 848
Introduction to Evolutionary Computation

Module 4, Lecture 21, Part 1
Parallel EC and Implementations

Erik D. Goodman, Executive Director
BEACON Center for the Study of Evolution in Action
Professor, ECE, ME, and CSE

Parallel EC Algorithms – *Independent of Hardware*

Three primary models: coarse-grain (island), fine-grain (cellular), and micro-grain (trivial)

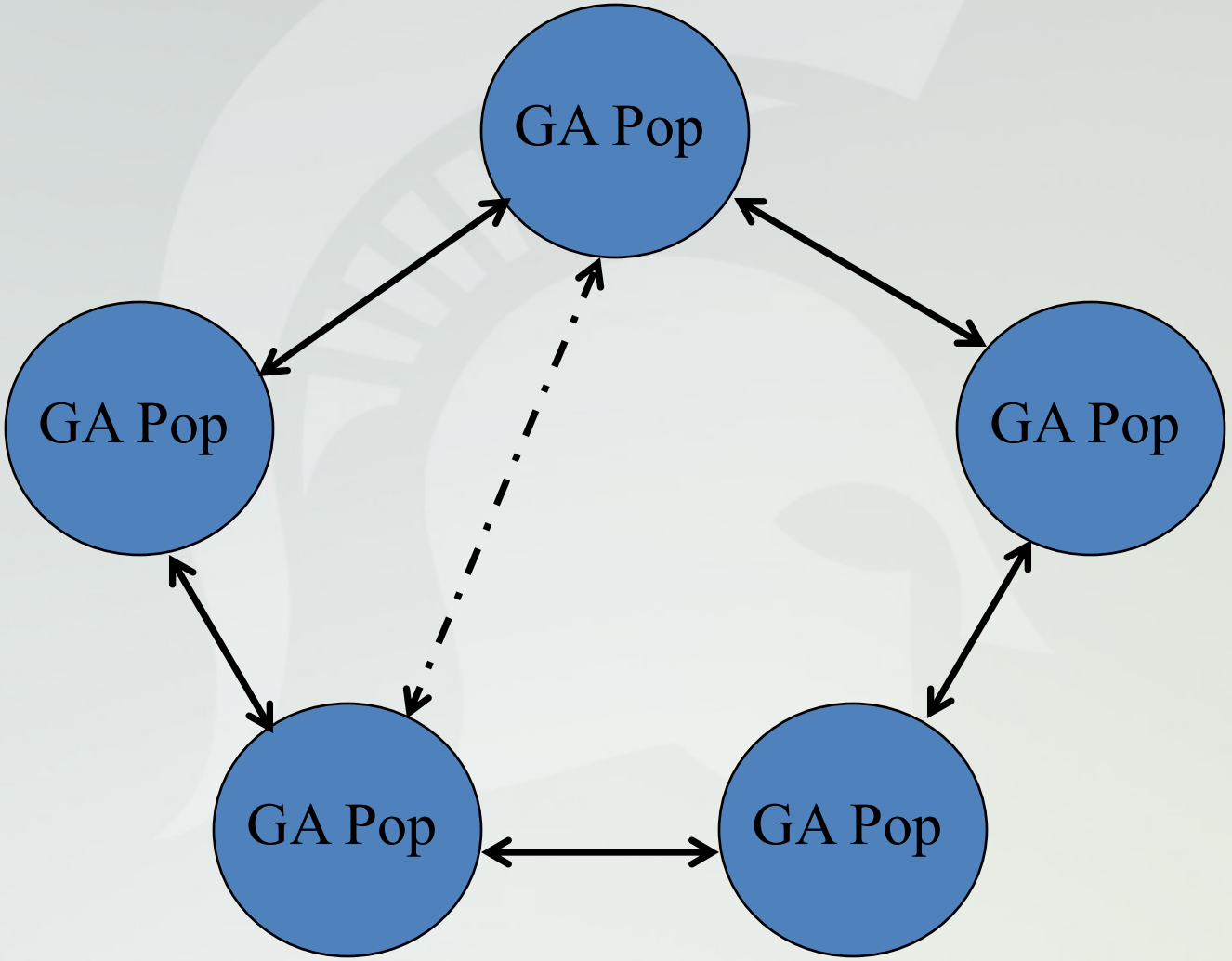
Trivial parallelism (not really a *parallel EC algorithm* – just a parallel *implementation* of a normal EC algorithm):

- passes out individuals to separate processors for evaluation (or runs lots of local tournaments, no master)
- still acts like one large population

Coarse-Grain (Island) Parallel GA

- N "independent" subpopulations, acting as if running in parallel (*timeshared* or *actually* on multiple processors)
- Occasionally, migrants go from one to another, in pre-specified patterns
- Strong capability for avoiding premature convergence while exploiting good individuals, if migration rates/patterns well chosen

Example: An Island Parallel GA



Migrant Selection Policy

Who should migrate?

- Best guy?
- One random guy?
- Best and some random guys?
- Guy very different from best of receiving subpop? ("incest reduction")
- If migrate in large % of population each generation, acts like one big population, but with extra replacements – could actually SPEED premature convergence

Migrant Replacement Policy

Who should a migrant replace?

- Random individual?
- Worst individual?
- Most similar individual (Hamming sense)
- Similar individual via crowding?

How Many Subpopulations? (Crude Rule of Thumb)

- How many total evaluations can you afford?
  - Total population size and number of generations and "generation gap" determine run time
- What should minimum subpopulation size be?
  - Smaller than 40-50 USUALLY spells trouble *for hard problems* – rapid convergence of subpop – 100-200+ better for some problems
- How many generations desired in each subpop?
- Calculate how many subpopulations you can afford

Example of Architecting an Island Parallel GA Run

- Say have expensive fitness function, difficult problem with 40 real-valued design variables to optimize, can afford to wait 5 days for a solution. Each evaluation takes 0.1 days on one core, and is not easily split among multiple cores.
- Assume you need at least 1000 evaluations to get a decent answer, based on prior work on related problems.
- So can do 50 evaluations on one core in 5 days, so need a minimum of 20 cores.
- Have a HPCC capability to access 100 cores, but paying for use according to number of core-days used.
- How would you structure the job?

What Are You Worrying About?

- Premature convergence?
- Enough generations to allow reliable combining of good building blocks?
- Global search vs. fine tuning? Behavioral novelty?
- What kind of GA would you use? Steady-state? Generational?
- Want 20 subpops of size 50? (then initial evaluations use up 1000 evaluations)
- Let's decide how many evaluations to burn up (of 1,000 total) on randomly initializing the subpops. Call that Init. Then if we have M subpops, each of size N, then M*N = Init.

More Thinking…

- For example, maybe Init = 100 evaluations, M=5, N=20.
- How many cores to assign to each subpop?
- Better be at least 4, to use up our minimum of 20 cores we know we need to finish in time
- If 4, then each subpop will do 4 evaluations in parallel. Good if evals/generation is a multiple of 4?
- Now, how many offspring to evaluate per generation (generation gap)? 1? 4? 8? 20? What is the effect?
- How often do we migrate, and which guy, and who does the migrant replace?

Still More Thinking…

- What about using more cores to do more evaluations in 5 days? What's the tradeoff?
- If you DID use more cores (say 100), what would you do with them? More generations of 5 subpops? More subpops? Larger subpop sizes? What about migration rates?