Organizing DSP Circuits on Pre-Built Hardware Using Evolutionary Algorithms

Fred Ma

Overview

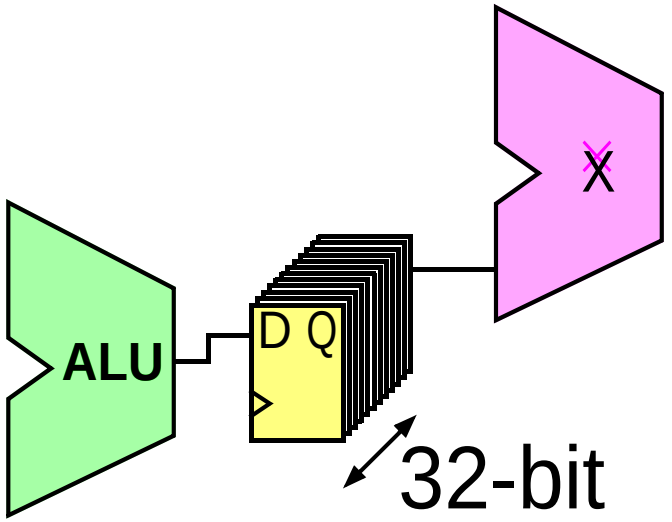
- Coarse grain *reconfigurable* hardware in telecom
- ***Example:*** Pre-built hardware and DSP circuit
- ***Problem:***
  Arranging the circuit elements on the hardware
- Solving the problem with genetic algorithms (GAs)
- Ensuring that solutions meet all physical constraints
- Results

Telecom Data Processing

- High throughput computations
- Regular, repetitive computations

} Hardware

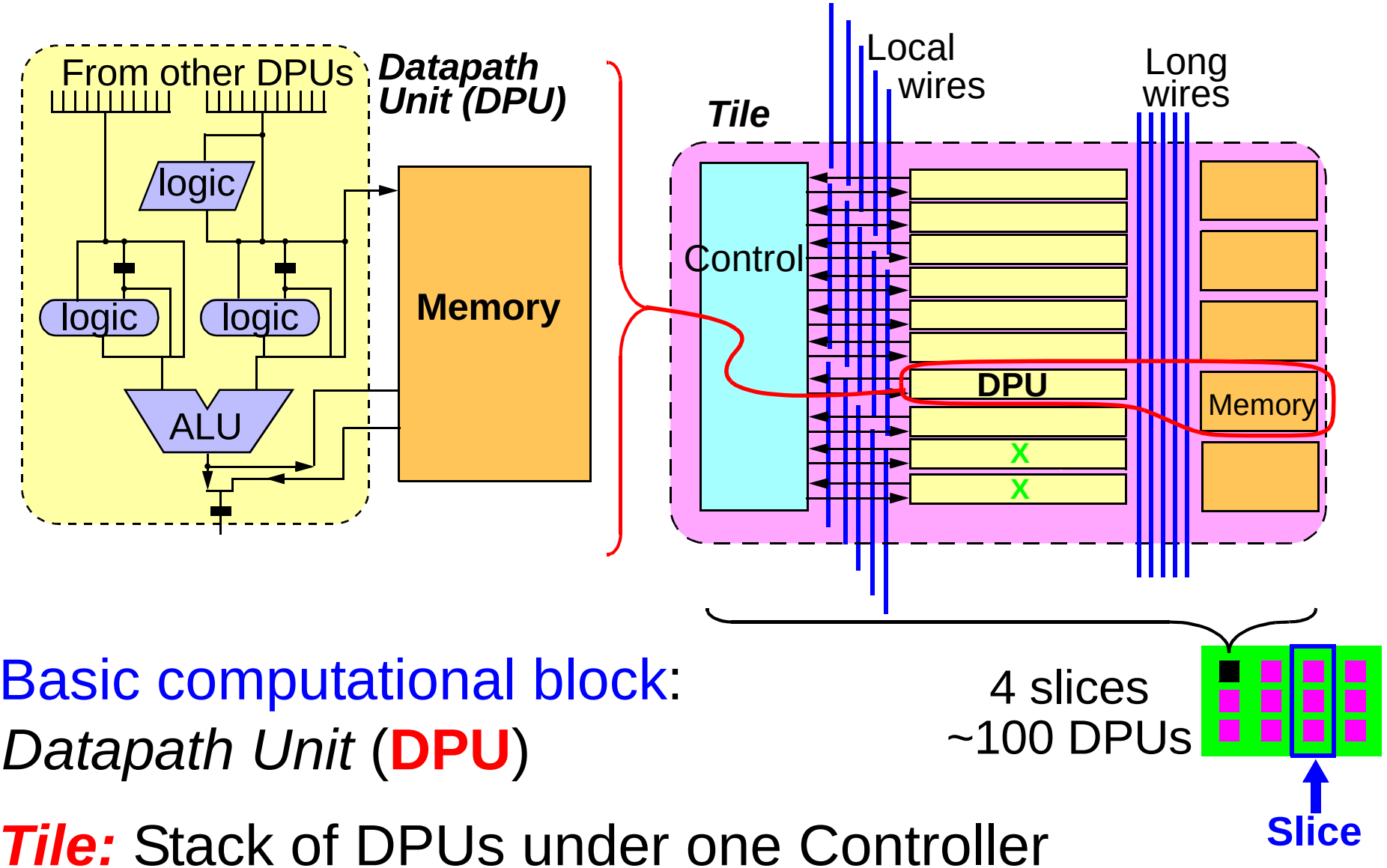
- Lots of numeric computation
  ⇒ ***Coarse grain*** hardware
  - *Swaths of bits* to represent numbers
  - Logic optimized for arithmetic



Why Reconfigurable?

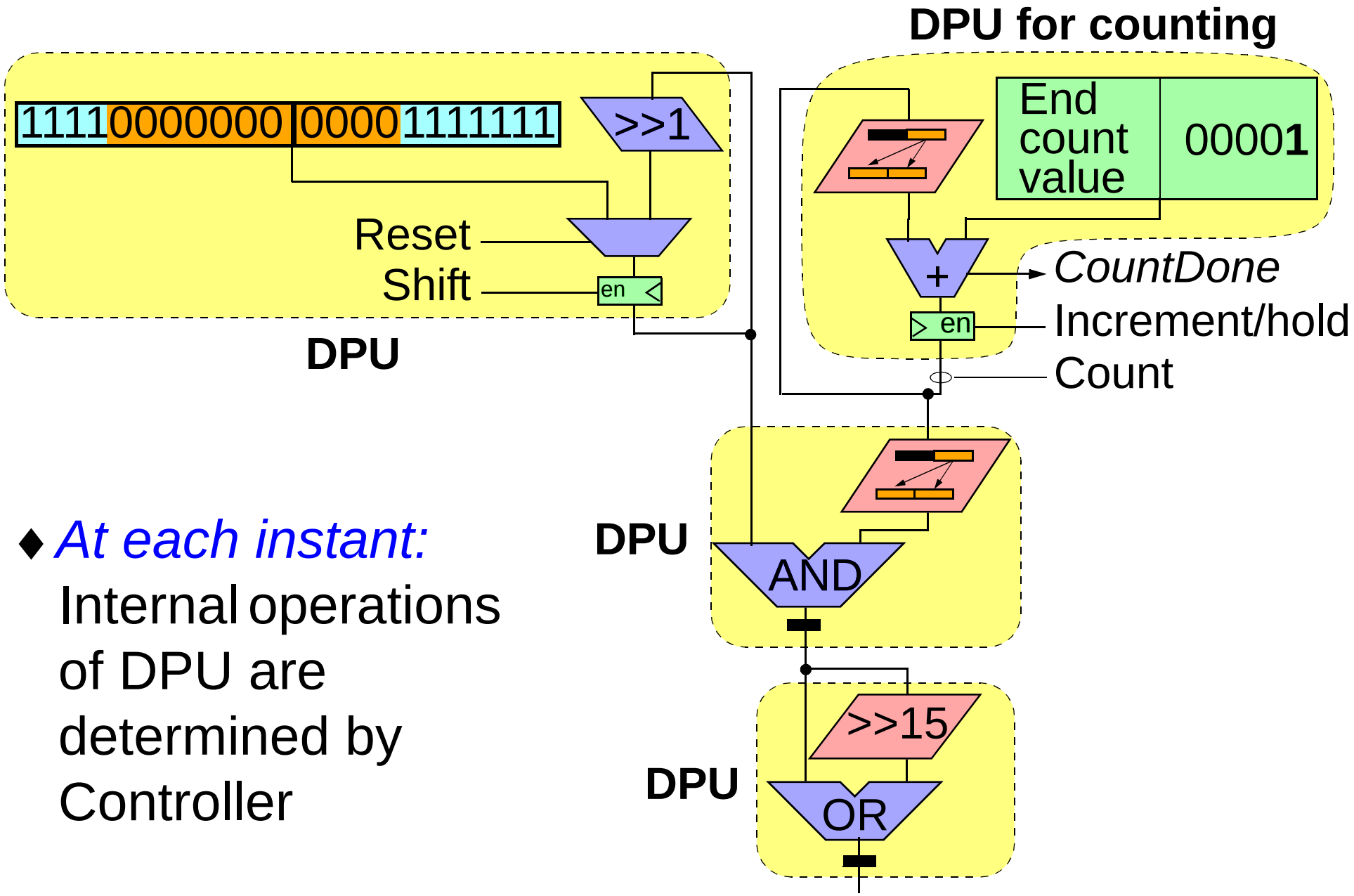
- Hardware speed, software configurability
- Avoid cost and delay of chip fabrication
- Ease of bug fixes, design revisions
  - Download new hardware to remote locations
- Adaptability to developing/evolving air interface standards

Chameleon Architecture



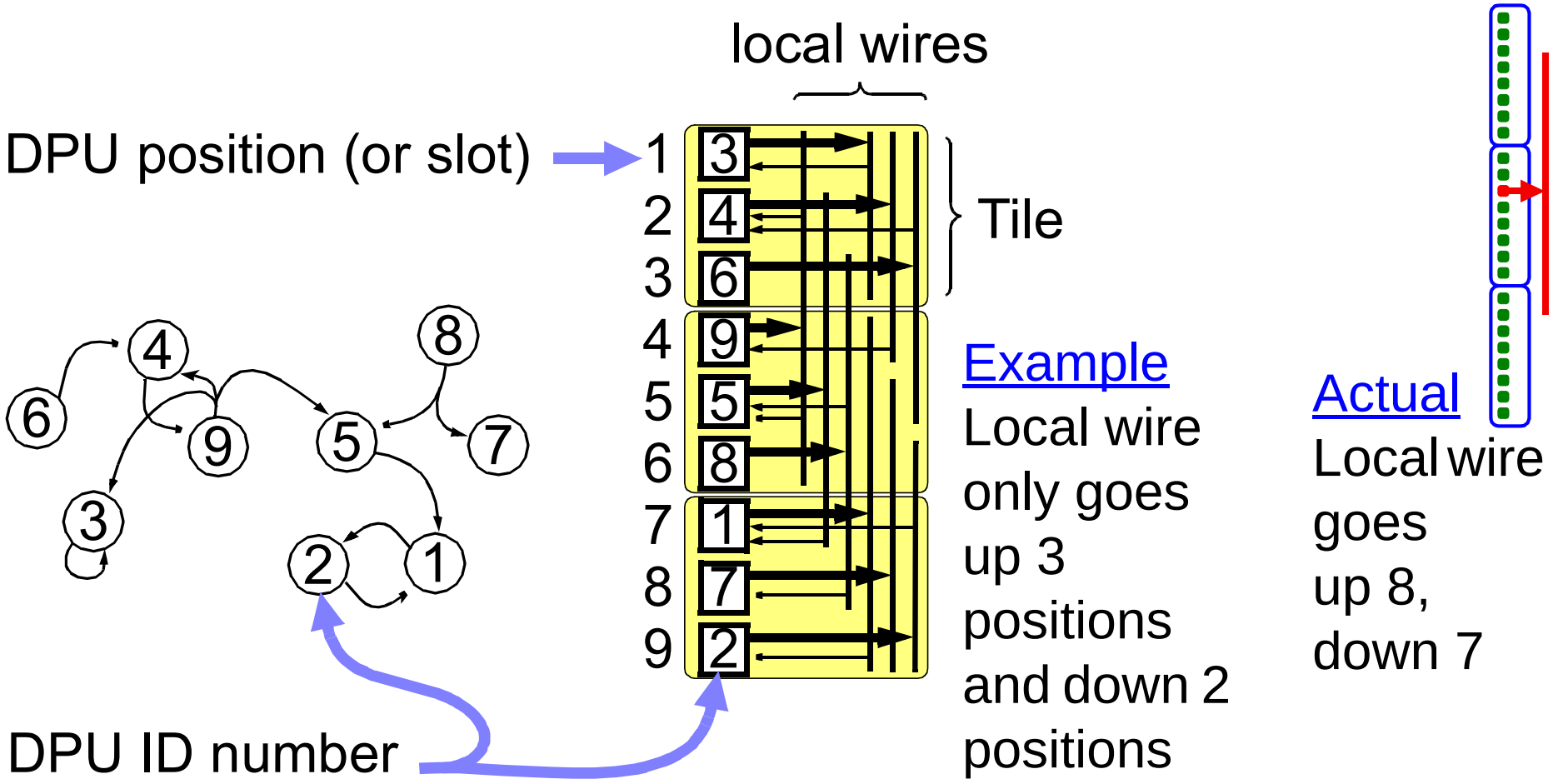
- Basic computational block: *Datapath Unit* (**DPU**)
- ***Tile:*** Stack of DPUs under one Controller
- ***Slice:*** Stack of Tiles

Example “Circuit” of DPUs



- *At each instant:*
  Internal operations of DPU are determined by Controller

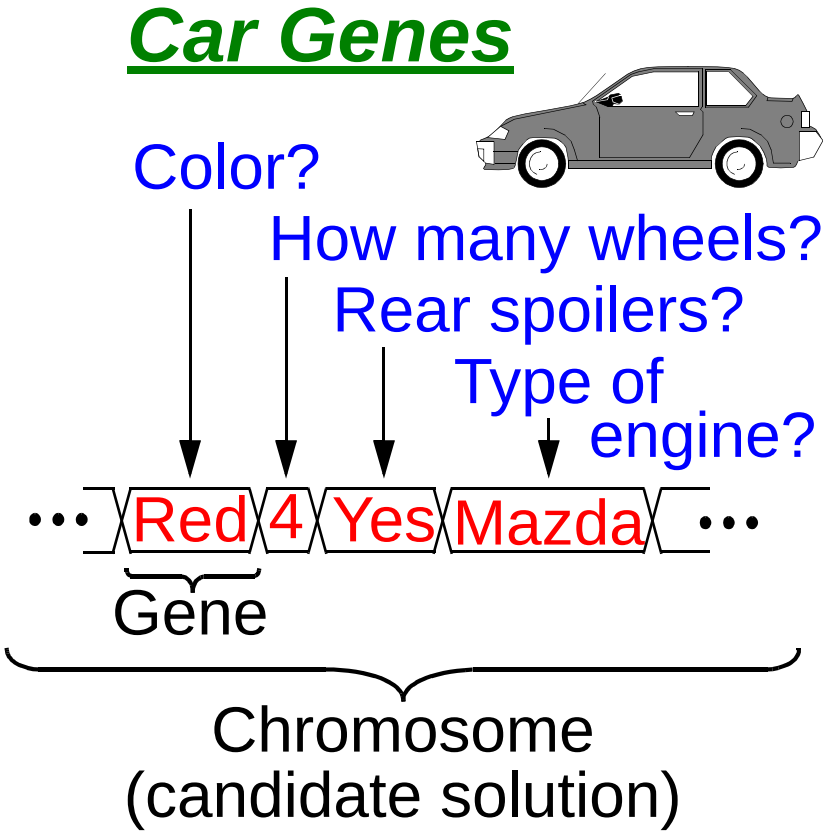
Placement of DPUs



- ***Simplified problem:*** Arrange DPUs onto array so that all connections can be made using local wires
- ***Actual problem:*** Additionally uses long wires, ***and*** ensures that location of each DPU supports the required functions *(tedious!)*

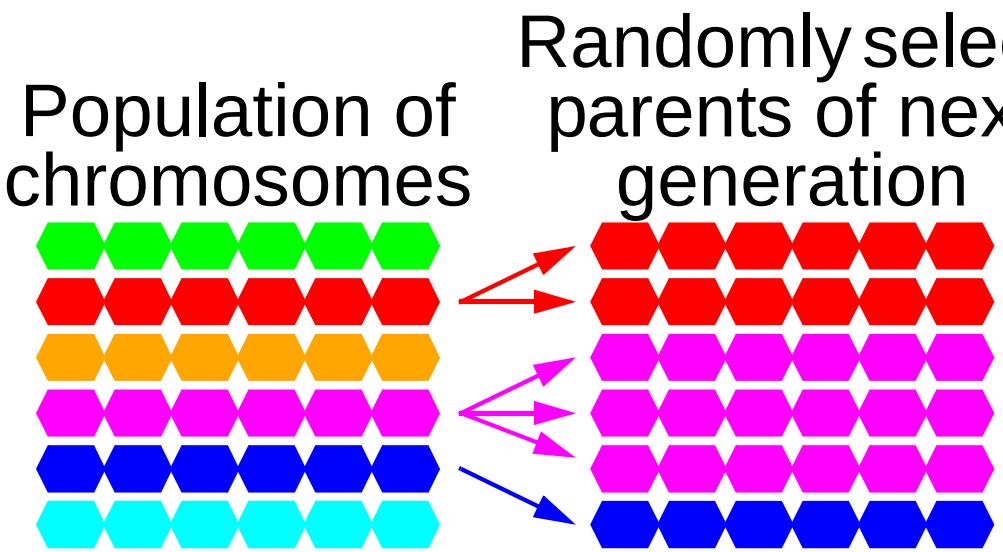
Genetic Algorithms (GAs): Educated Trial and Error

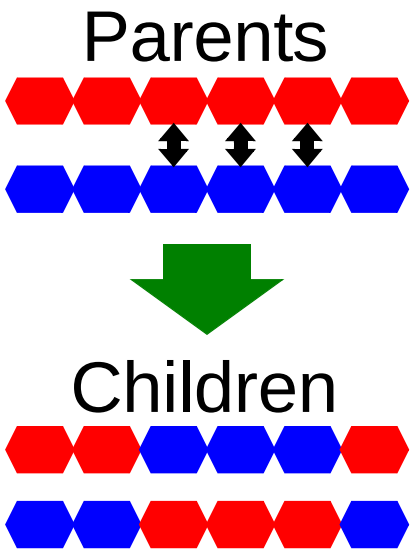
- For combinatorial problems (resource allocation, scheduling)
- *Encode candidate solutions* as a strings of parameter values *(chromosomes)*
- Start with a large *population of solutions*, *combine good ones* to create better solutions
- ***Fitness:*** A chromosome's measure of goodness



3-wheel car
not very good

Improving the Solutions: Evolution





Selection

Give higher probability of selection to high fitnesses

Crossover

Pair off selected parents and swap genes

- Population members selectively replaced by promising offspring in each generation
- Population's average fitness improves

Pros *(& Cons)* of GAs

- GAs are *heuristic* and *stochastic* (not true optimizer)

  We don't care about optimization!

  We just want a *any arrangment* of DPUs that satisfies all requirements.

- True optimization algorithms are ***highly*** dependent on problem details

  Slight change in DPU array architecture can completely change the problem

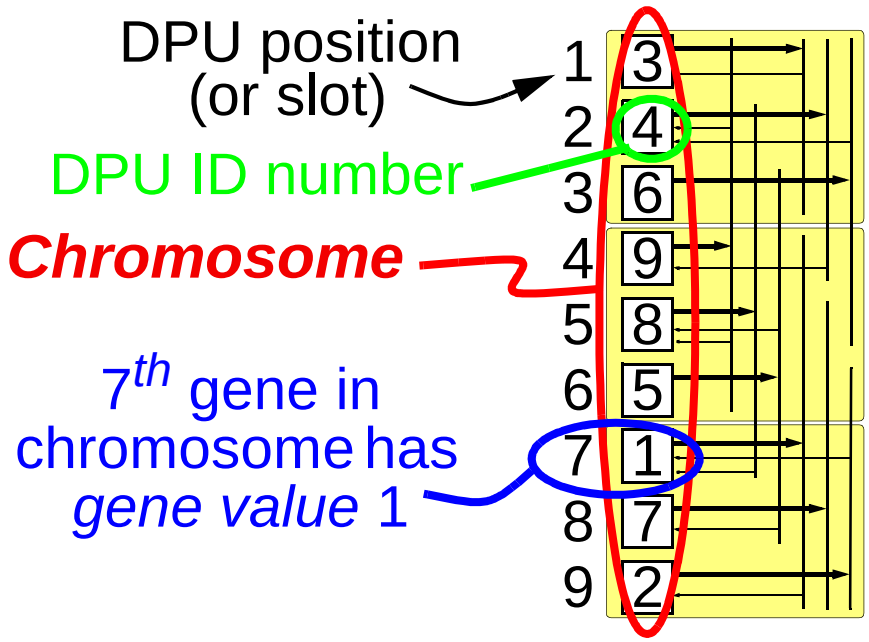
  GA chromosomes are *easy to encode* and strange requirements are *easy to enforce*.

- ***But .....***

  As a statistical search method, GAs can computationally demanding

Genetic Placement of DPUs

- Valid solutions must meet *many constraints*
- Newly generated chromosomes must be *permutations of the DPUs*
- Some DPUs need to be in *specific tiles* because of memory or control

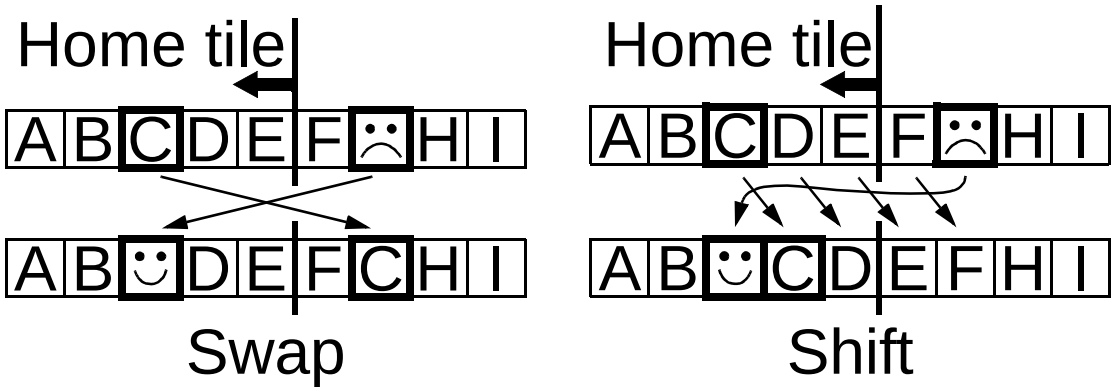


- Some DPUs have to be in either *even or odd positions* or special positions containing *multipliers*

⇒ Many solutions will NOT be valid

Enforcing Positional Constraints

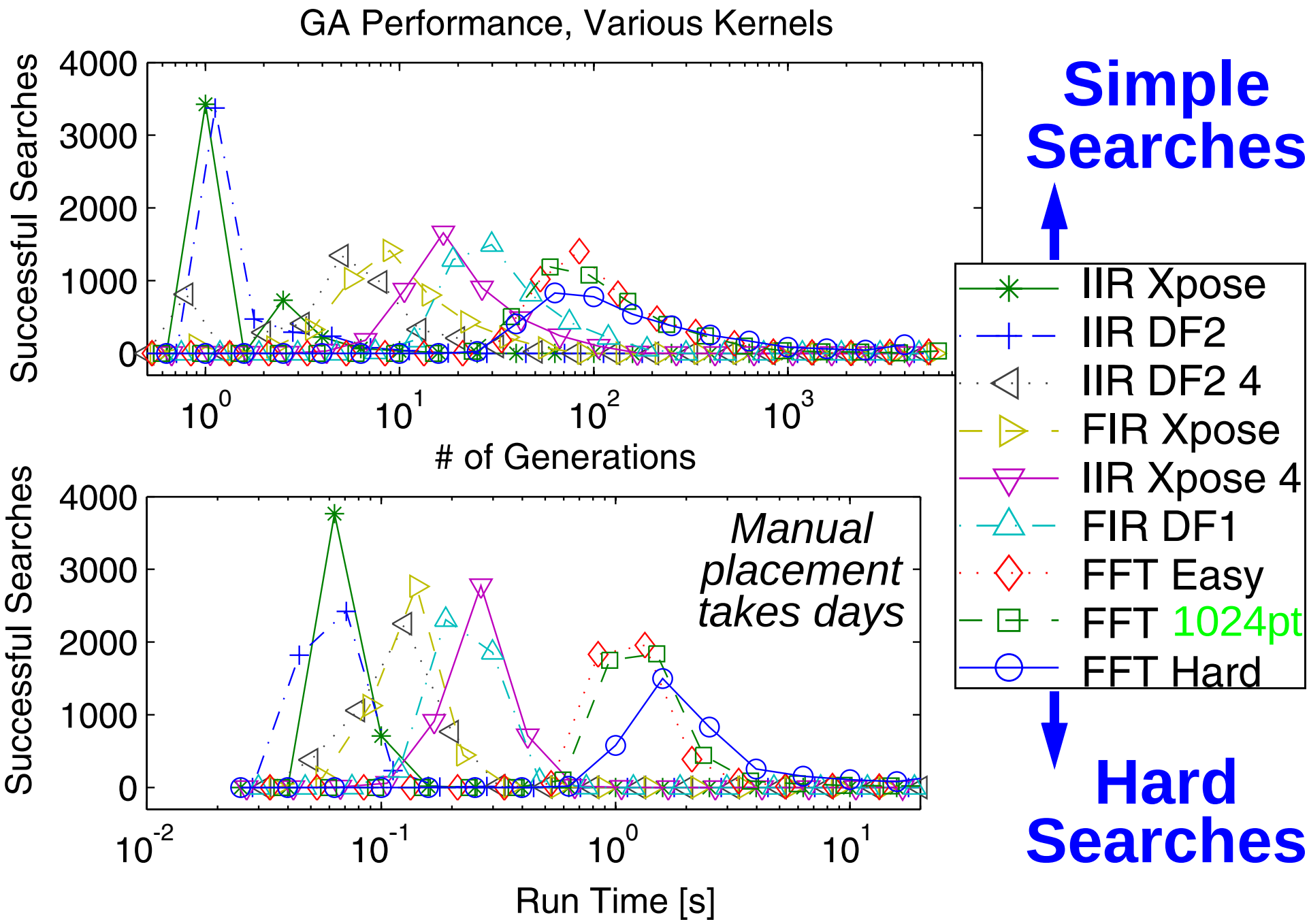
- During fitness evaluation, chromosomes undergo problem-specific *heuristic repairs for all constraint violations*



- *Non-repairable violations* cause a *penalization in fitness* to discourage their proliferation
- Some *violation types are avoidable* by *clever encoding* of the problem parameters into chromosomes

Results

Effectiveness of GA Placement



Experimental Findings

- *50% population change* per generation is best
- *30%~50%* of the new solutions should be created from *crossover*
- *50%~70%* should be from ***mutation***

  Random variation of existing population members to guard against inbreeding

- Computational effort correlates well with other measures of circuit complexity
- *Significant reduction in search times* from *heuristic repair* of constraint violations
- Search times comparable to *simulated annealing*

Conclusions

- Flexibility of problem encoding and solution evaluation

  GAs easily adapt to diverse constraints
  from different reconfigurable hardware

- Combining GA search with repair is a crucial strategy

  Hybridize GA with “local search”

- Traditional role of mutation as minor background process can be far from optimal

- The developed GA places real-world circuits in seconds

  Good enough for interactive CAD

Further Details

- F. Ma, J. P. Knight, and C. Plett, “Physical resource binding for a coarse grain reconfigurable array using evolutionary algorithms”, *IEEE Transactions on Very Large Scale Integrated Systems*, (accepted).
- F. Ma, J. P. Knight, and C. Plett, “Physical resource binding for a coarse grain reconfigurable array”, *Engineering of Reconfigurable Systems and Algorithms*, June 21-24 2004.
- F. Ma, J. P. Knight, and C. Plett, “Reconfigurable logic design case”, *SPIE Conf. on Reconfigurable Technology: FPGAs and Reconfigurable Processors for Computing and Communications (part of ITCom 2002)*, v. 4867, pp. 113–126, July 30 2002.

The End

Reference Slides

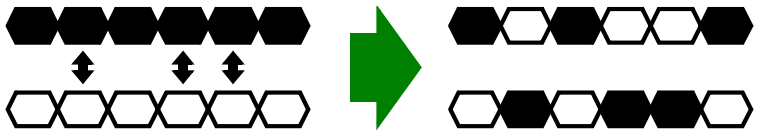
How Hard is the Problem?

- 10^{28} possible orderings of DPUs
- Heuristic search difficulty
  - Depends on *density* of valid solutions in search space
- ***Random exhaustive search***
  - Tried billions of combinations over a week
  - Couldn’t find solution for FFT
  - Enabling repair of violations didn’t help

GA Parameters

(from extensive experimentation)

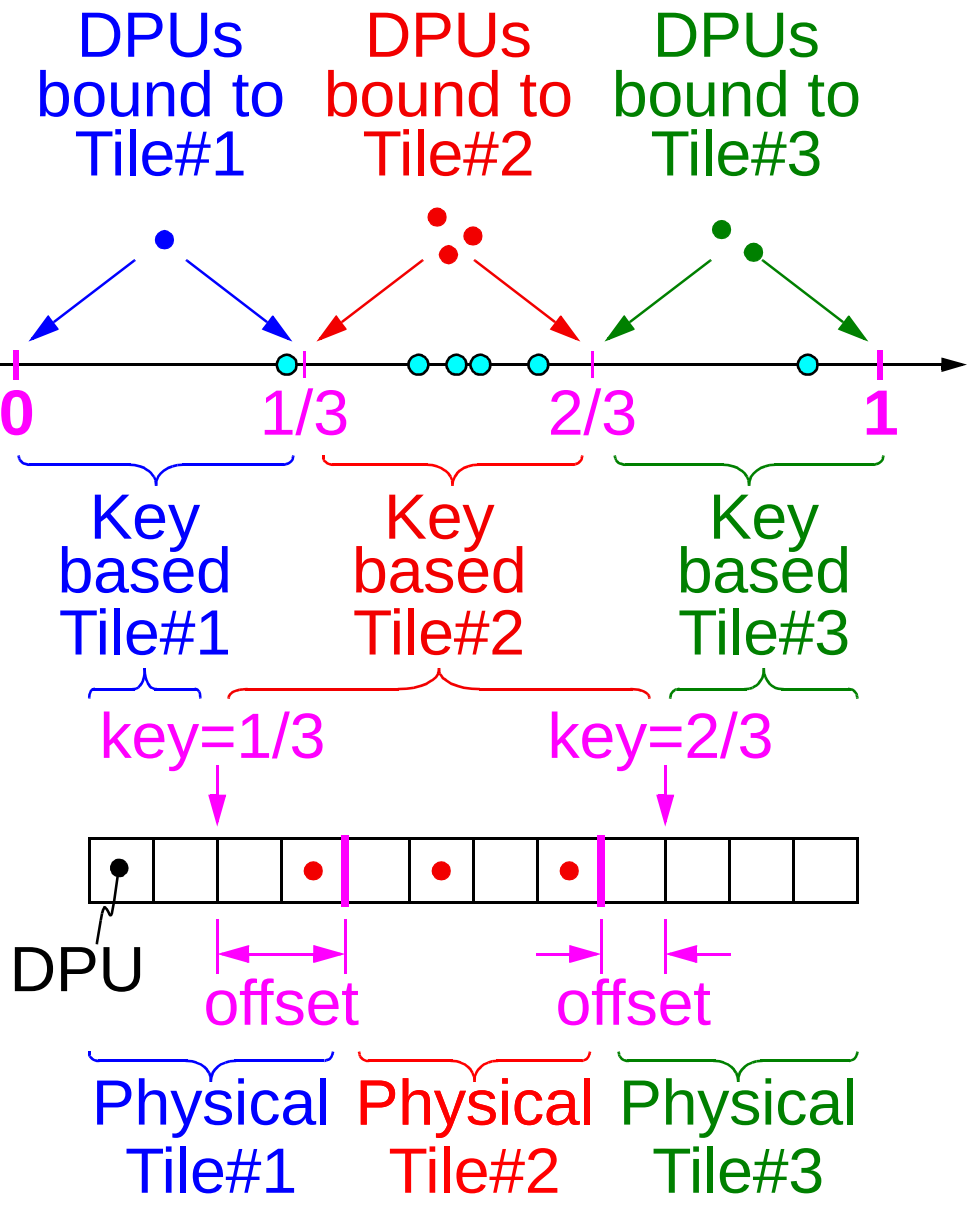
- Random Keys encoding:
  - Each DPU assigned a random real “key”
  - Get DPU order by sorting the keys
- Uniform crossover: Randomly choose genes to swap



- Quadratic penalty for wires exceeding 8 DPU positions
- Ranked fitness to avoid scaling problems
  - Rank of genome after sorting population by descending cost
- 1~2 global vertical wires driven from each tile
- Mutate general population members for diversity

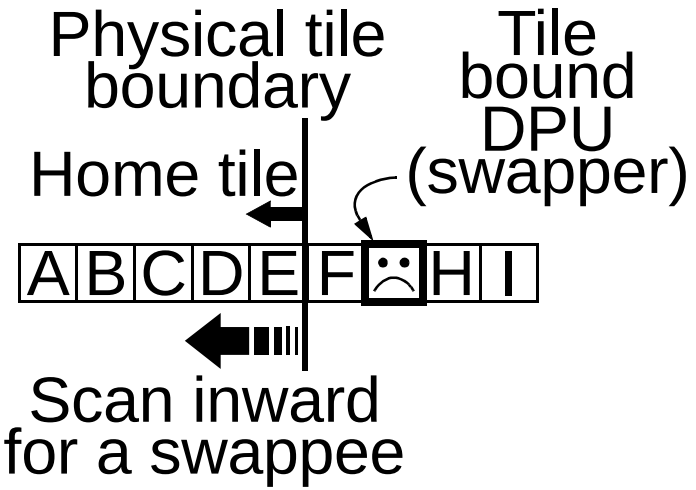
DPUs Belonging in Specific Tiles

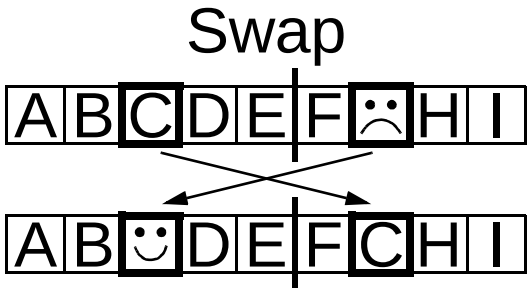
- Equal size subranges define *key based* tiles
- *Physical* tiles defined by counting 4 DPUs/tile
- **The problem:**
  Key based tile edges not the same as physical tile edges
- **Solution:**
  Penalize misaligned tile edges
  - Treat offset as an overlength wire
  - Misalignments minimized by evolution

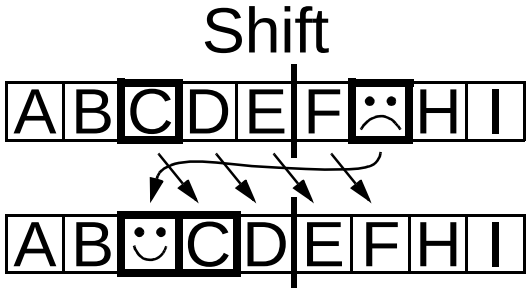


Fixing Stubborn Tile Binding Violations

- Aggressive circuit design: Hard to meet all constraints
- Seek *simple* re-ordering to fix violation
- Swap errant DPU into home tile
  - Don’t create/worsen a tile violation
- Multiple passes
  - Each pass tries to fix all tile violations

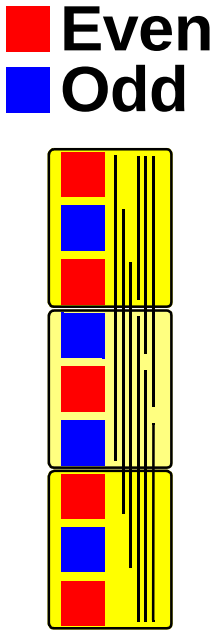




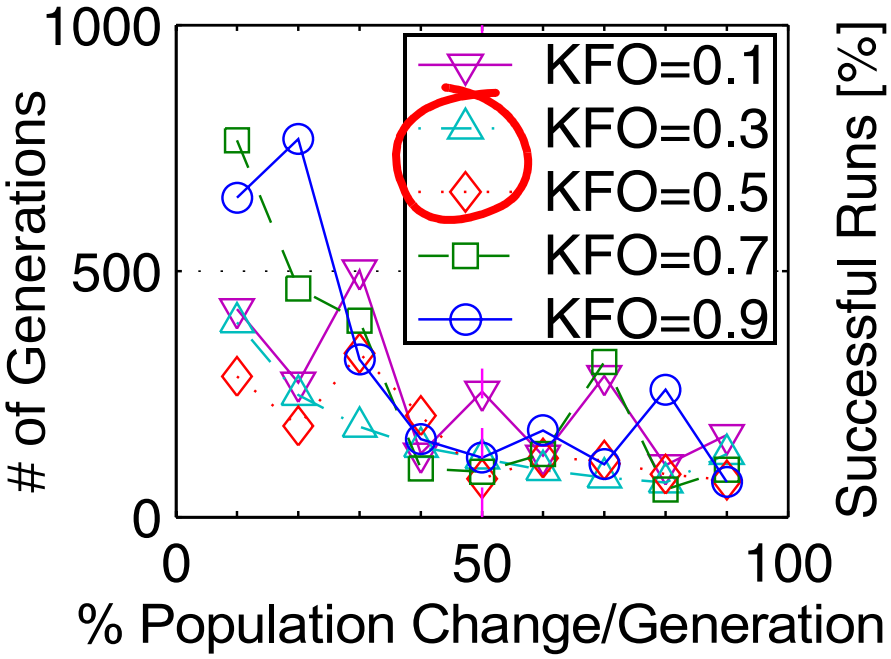


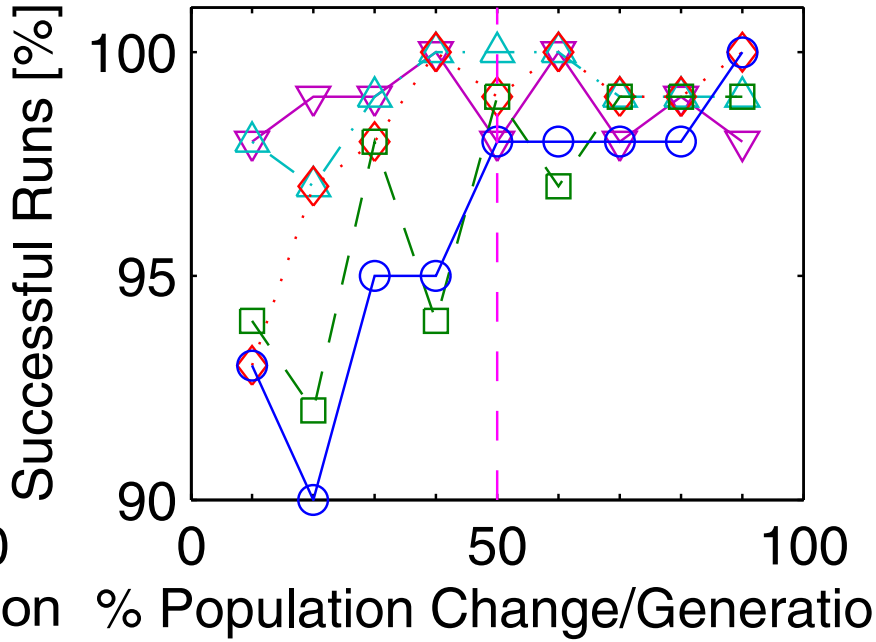
Even/Odd Position Constraints

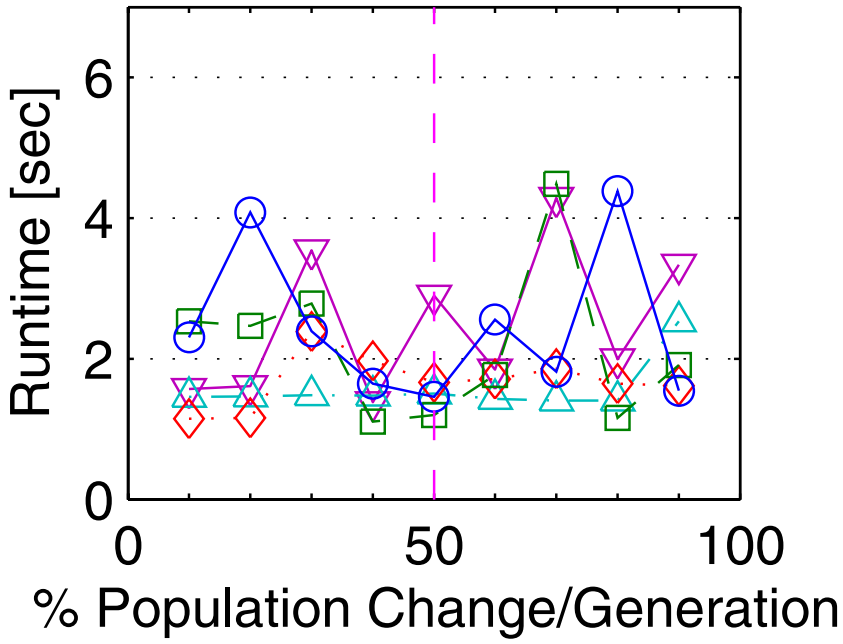
- Fine positional constraint relies completely on local search for compliance
- Save area by interleaving DPUs with complementary functions
- A polarity violating DPU checks immediate neighbours for swapping
- *Small fine-tuning* movements



Identify Good Search Conditions (FFT)







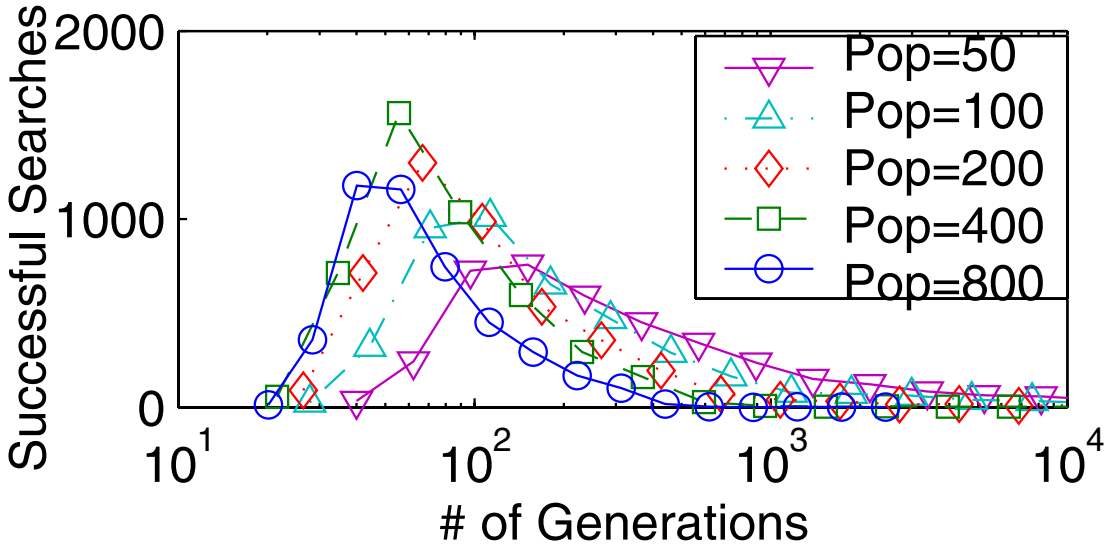
KFO:

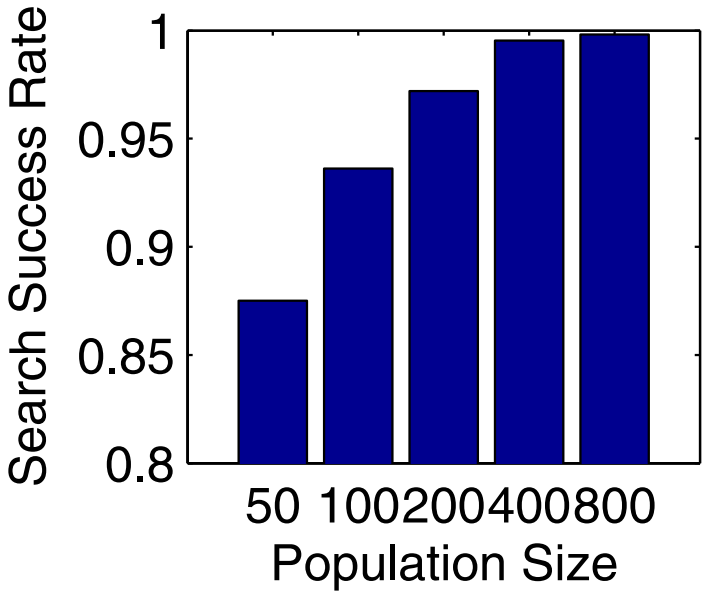
***K*ids *F*raction *O*ffspring**

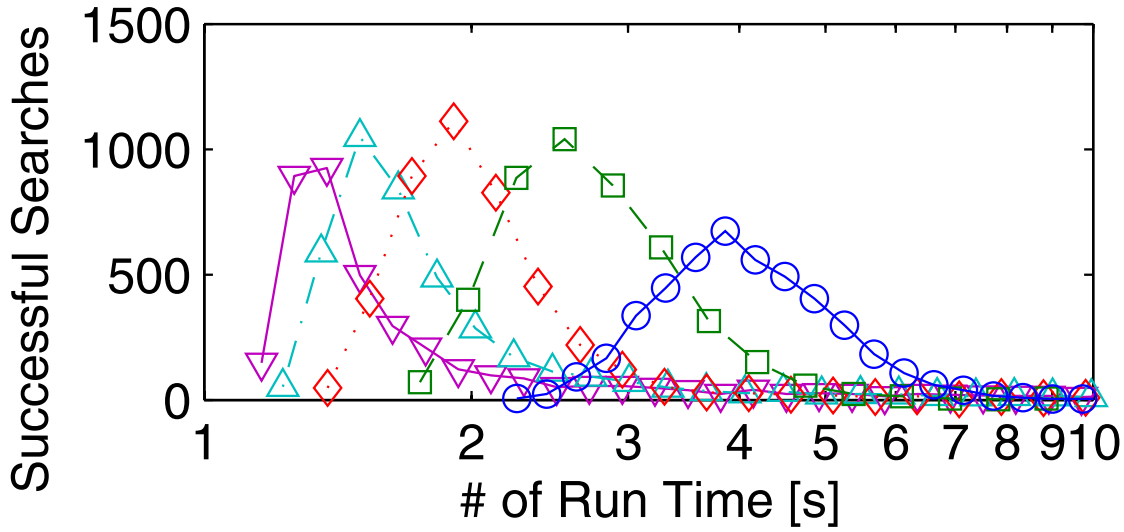
- The fraction of kids made up of offspring (rather than mutants)
- Indicates greedy search

Effect of Population Size

(FFT Kernel)



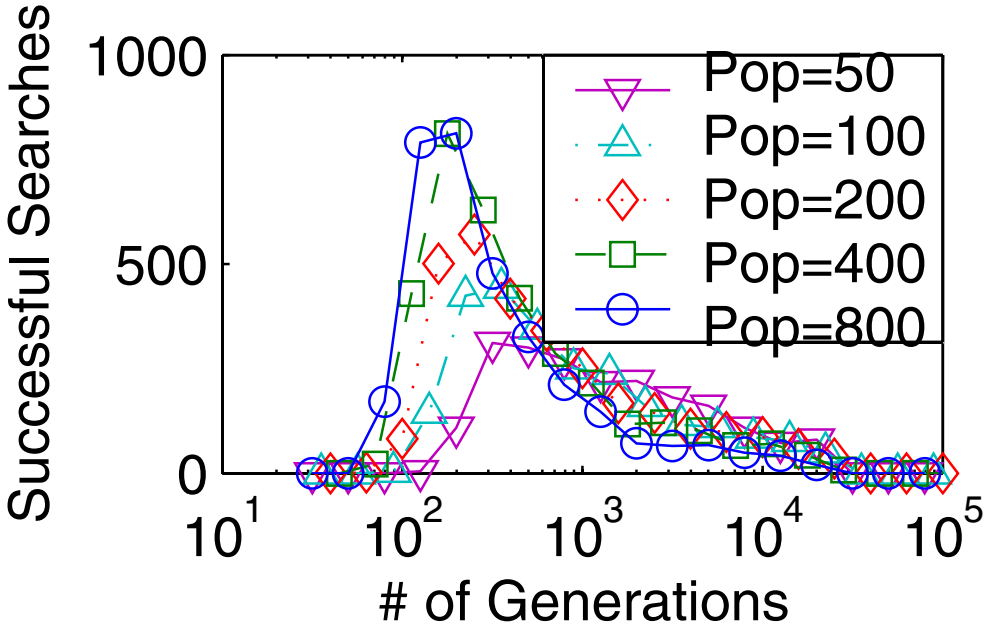


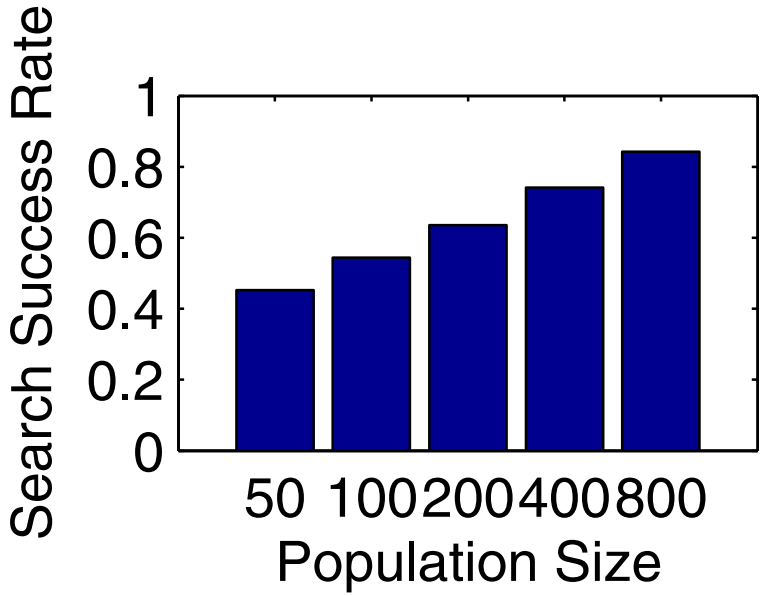


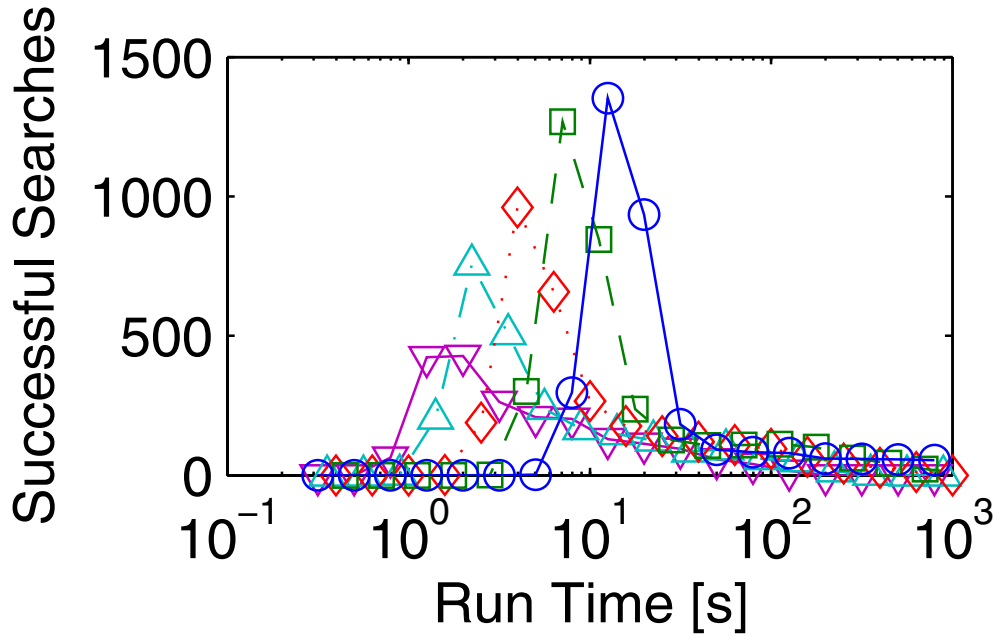
Increasing Size:

- Fewer generations
- Longer execution
- Higher success rate

Back-to-back FFT/IFFT



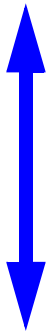




- Double netlist size, record length, and DPUs/slice
- Local wires still span 8 DPUs
- 5 global wires/tile vs. 1
- 5x generations
  50% more runtime
- Much reduced success rate

Circuit Complexities

Easy



Hard

Kernel	DPUs	MULs	DPUs+ MULs	Nets	Driven Ports	Tiles
IIR Xpose	7(2)	5	12	11	15	1
IIR DF-II	8(2)	5	13	12	15	1
IIR DF-II x4	14(8)	5	19	15	21	4
FIR Xpose	10(2)	5	15	14	22	1
IIR Xpose x4	13(8)	5	18	14	27	4
FIR DF-I	19(2)	5	24	23	31	1
FFT Easy	20(2)	6	26	24	50	3
FFT	20(2)	6	26	24	50	3
FFT Hard	20(2)	6	26	24	50	3

GAs Compared to Simulated Annealing