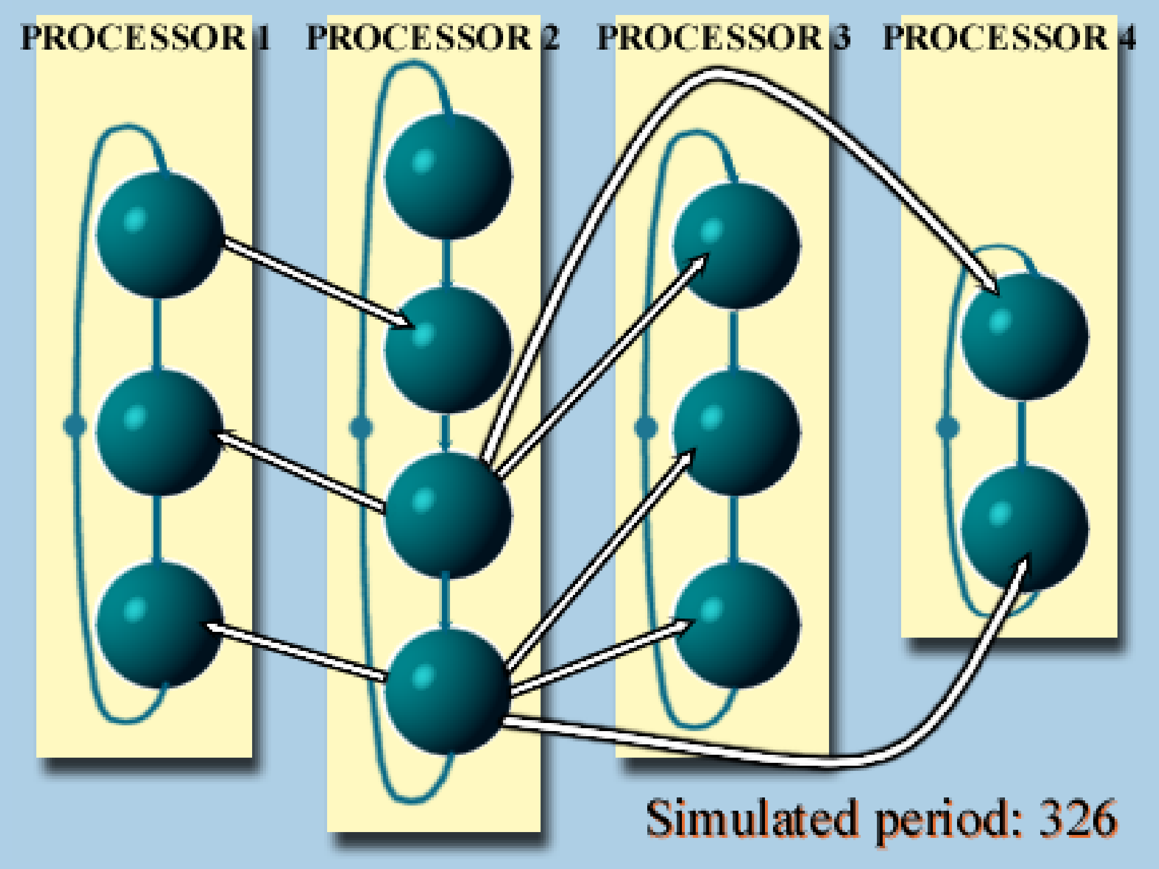
Genetic Resynchronization of Multi-Processor Embedded Systems

DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

J. Bonanno, T. Sheng, K. Kimes
S. Bhattacharyya

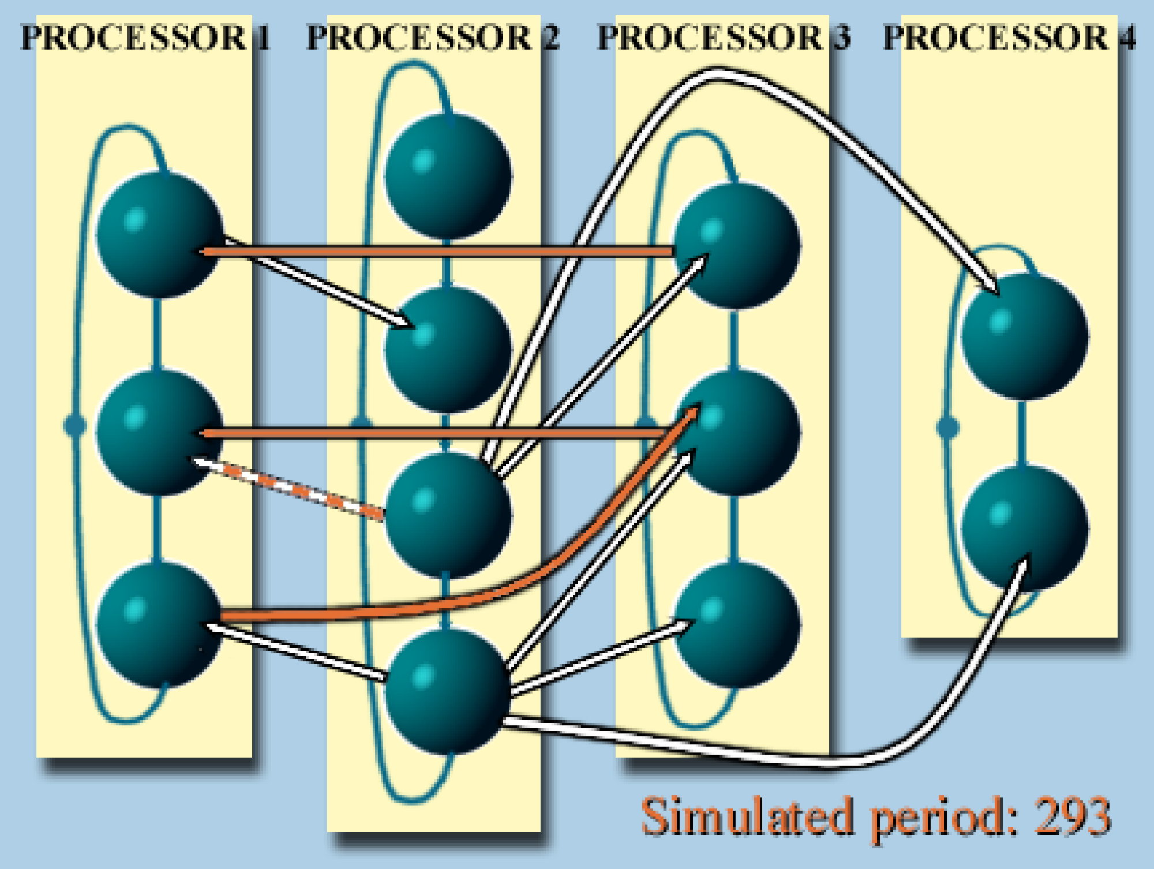


A randomly generated schedule, a candidate for resynchronization. The white edges represent synchronizations, while the blue ones show processor groupings.

Problem Description

Due to the availability of low cost digital signal processing chips, the consumer electronics field has experienced many breakthroughs in cost and efficiency. Squeezing the most out of these systems, which often have more than one processor, has become critical to remain competitive. Significant improvement in these types of systems can be realized by eliminating unnecessary inter-processor communication.

Before data can be transferred between two processors, they both must be ready to send or receive that data, respectively. This handshaking is referred to as synchronization. It is possible, however, to force this handshaking to only occur at specific points, which ensure that the processors will be synchronized at later times, thereby reducing the total synchronization needed and improving performance. The process of attempting to determine an optimal placement of synchronizations is called resynchronization. Finding an efficient way to do this is the fundamental problem we are trying to solve.



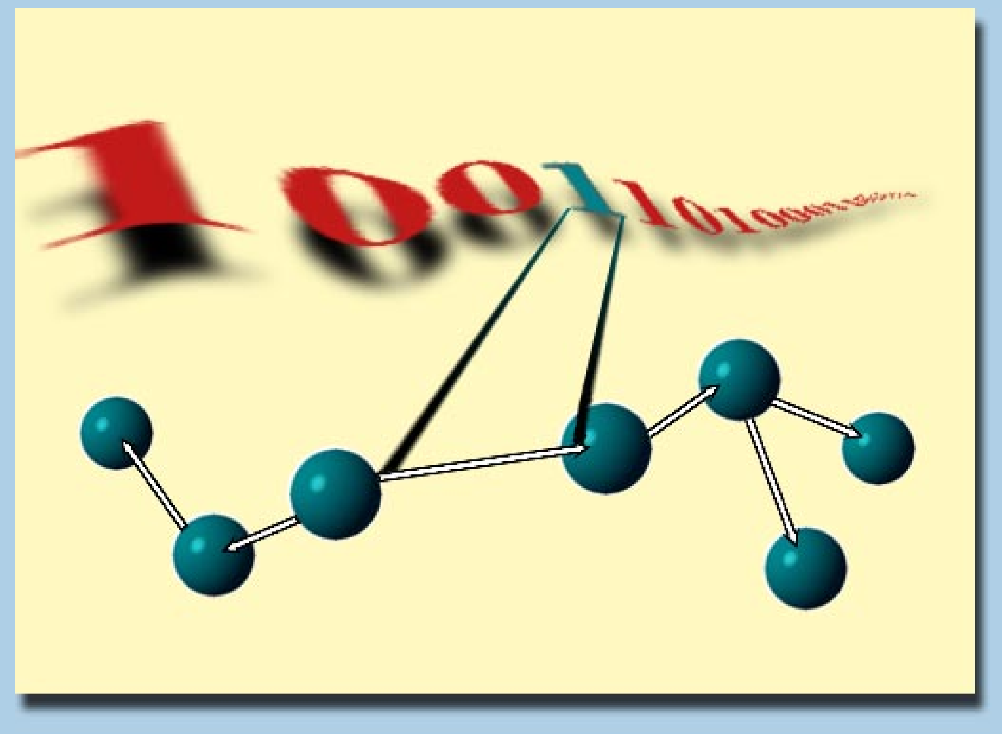
A resynchronized graph. The orange edges are new synchronizations. The dotted orange edge is now redundant and can be removed. The simulated period has decreased.

Simulator

In order to accurately analyze the performance of the algorithms we developed, we needed a detailed representation and simulation of the system. We developed a number of tools, which enable the user to examine the system and weigh the benefits of the resynchronization. The resulting suite of applications should retain its usefulness far beyond the scope of this project.

Strategy Overview

- Resynchronization is an NP-complete (computationally intractable) optimization problem
- Therefore, we must identify heuristic algorithms
- Genetic algorithms are heuristics that often work well on optimization problems
- Genetic algorithms use Darwinian natural selection to *evolve* a good solution
- Solutions must be modeled by chromosomes (often a string of 1's and 0's)
- Each chromosome must have a measure of *fitness*, its value as a solution
- Genetic operations (*recombination, mutation, immigration*) are carried out on a population of chromosomes
- The best members of the population will be *selected* for the next generation
- After several generations, the population should contain several very good solutions



Each position in a chromosome represents a possible edge to be added to the graph.

Strategy Implementation

Our algorithm takes an input schedule and computes the set of possibly beneficial edges to add to this graph. These edges are those which:

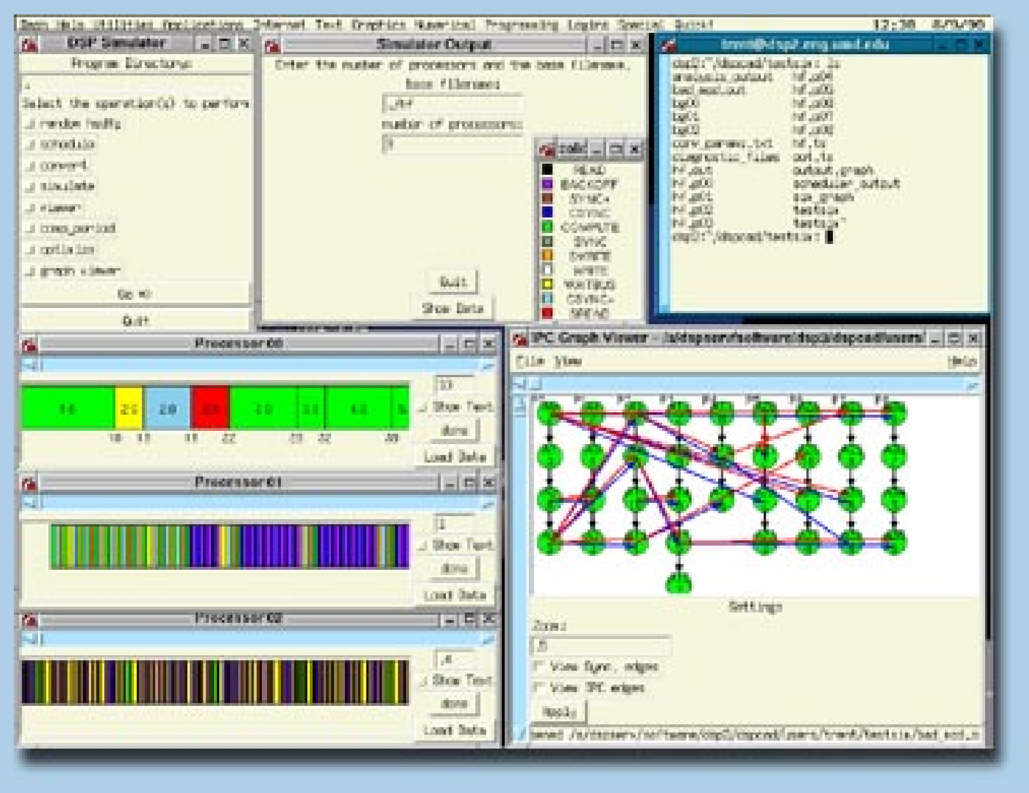
- do not increase the maximum cycle mean, an estimate of performance
- do not cause deadlock
- do not already exist

In our implementation the following have special meaning:

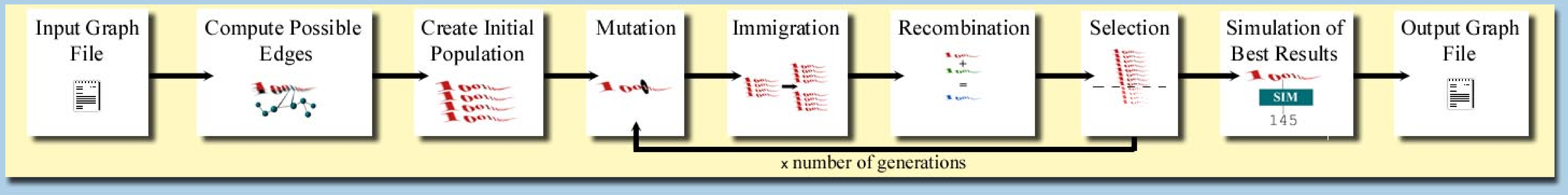
- Chromosome - a binary string where each bit specifies whether or not to add a particular edge of the set computed earlier.
- Recombination - crossover, two combining chromosomes (parents) are split into two segments each, the child takes one portion from each of its parents.
- Fitness - a function of the change in maximum cycle mean from the original graph, and a change in the number of interprocessor synchronizations.

Features

- Visualization of processor activity
- Visualization of input graphs
- Easy to use graphical front-end (written in TCL-TK)
- Period and iteration analysis
- Cross-platform portability of simulation engine (written in C++)
- Detailed modeling of shared bus access
- Detailed modeling of synchronization protocols (feedforward vs. feedback)



A screenshot from the simulation tools. In the lower left is a view of processor activity. The lower right shows a view of the input graph.



The program flow for our genetic resynchonization algorithm.