A Genetic Algorithm for Scheduling Splittable Tasks with Precedence Constraints
Yuanliang Gao and Sheung-Hung Poon

Background
- Task graph scheduling on multicore processors
- Combinatorial optimization
- Objective: Efficient usage of time
- Reduce the length of the schedule (makespan)
- Scenarios:
  - Scientific computing, cloud computing, other high-performance software

Highlights
- Developed a genetic algorithm for scheduling task graphs with splittable tasks where the splitting overhead is considered
- Previous work: either do not consider splitting overhead, or do not provide algorithmic way to decide the splitting
- Compared output quality with MIQP
- Our genetic algorithm:
  - Autonomously decides the points of splitting
  - Significantly faster than MIQP
  - Produces competitive solutions
  - Has potential to handle large input

Future Directions
- Extend the algorithm to handle communication time/heterogeneity
- Investigate the relationships between:
  - Genetic operator performance and task graph topology
  - Reproduction strategy and convergence speed
- Parallel genetic algorithms
- Linearize the MIQP model
- Develop heuristic methods for real-time environments

Input and Output
- Individual Representation:
  - Fractions of each subtask
  - Scheduling order of all subtasks from all tasks
- Population Initialization:
  - Fractions are randomly initialized
  - Scheduling order is a random permutation of consecutive integers
- Fitness Evaluation: First-fit list scheduler
- Precedence constraints handled here
- Subtask Merging Heuristic: Merge subtasks of a task scheduled on the same processor core to save splitting overhead
- Survivor Selection: Round-robin tournament

Experiment 1
Schedule Makespan
![Graph showing schedule makespan for different input sizes and algorithms (GA and MIQP)].

Experiment 2
Schedule Makespan (50-100)
![Graph showing schedule makespan for different input sizes and algorithms (GA and MIQP)].

Experiment 2
Schedule Makespan (300-1250)
![Graph showing schedule makespan for different input sizes and algorithms (GA and MIQP)].

Fig. 3. The number of iterations before the termination of the genetic algorithm. The gray line on the bottom shows parameter c set in the termination condition. The dotted line is an exponential regression based on the average of the results for each problem instance.

Fig. 5. The lengths of the schedules produced by MIQP and the GA for input sizes of 50 and 100. The categorical labels in the form of c/p denote the input file size. The bars for the GA show the average of 10 runs. The error bars show the min-to-max ranges.

Fig. 6. The lengths of the schedules produced by the GA for the rest input task graphs. The bars for the GA show the average of 10 runs. The error bars show the min-to-max ranges.

Fig. 4. The CPU time spent by the genetic algorithm in each run. The dotted line is an exponential regression of the average of results for each problem instance.

Fig. 7. The numbers of total iterations before the termination of the genetic algorithm during each run. The vertical axis uses a base-10 logarithmic scale. The dotted line is a logarithmic regression of the average number of iterations for each input file. The gray line on the bottom shows the parameter c used in the termination condition.

Fig. 8. The CPU time consumed in each run of the genetic algorithm. Likewise, the vertical axis uses a base-10 logarithmic scale. The dotted line is a logarithmic regression of the average consumed CPU time.