



Farm Generator

Seyed Ali Tabatabaee
Parallel Algorithms and Architectures



Introduction

A load-balanced farm consists of a single central process which distributes tasks, several processes that work as servers that solve their given tasks and another process which handles the results outputted by servers.



Goals of the Farm


- » Problem is solved by solving subproblems on servers
- » Workload is balanced among the processes
- » Processes are kept busy as much as possible
- » **The runtime for the whole problem is minimized**



Objective of the Project

- » Write a Python program which gets as input a config file and three functions written in C
 - ◇ `generate_task()`
 - ◇ `process_task()`
 - ◇ `process_result()`

and creates a C source code for a farm using MPI libraries that solves the problem given by the user





Initial Milestones

1. Developing a simple farm using MPI in C
2. Writing a Python code to generate the code of the simple farm based on simple config parameters
3. Adding more complexity to the farm (more efficient scheduling algorithms & communications)
4. Recognizing more config params (more control to user)
5. Developing real-world examples for testing farm



1- Developing a simple farm

- » Achieved on the first phase
- » Easy 😎



2- Writing code generator in Python

- » Achieved on the second phase
- » Using **Cog**
 - ◇ A simple code generation tool written in Python
 - ◇ Lets you use pieces of Python code as generators in your source files
 - ◇ Finds chunks of Python code embedded in files, executes the Python code, and inserts its output back into the original file.



3- Adding more complexity to farm

- » Scheduling Algorithm [phase 1 & 2]
 - ◇ Push-Based: center pushes task to servers based on information it has
 - ◇ Pull-Based: servers ask center for new task
 - ◇ Combination: combines pull and push (pull is priority)
 - ◇ Combination Then Pull: starts with the combination but only uses pull in final stages



3- Adding more complexity to farm

- » Consider more generic data structure for tasks and results and handle their communication [phase 2]
- » Add analysis structure and possibility for receiver to send feedbacks from processed results to center [phase 2]



4- Recognizing more config params

- » Achieved on the second phase
 - ◇ Choosing scheduling algorithm between the 4 provided options
 - ◇ Determining if receiver sends analysis to center
 - ◇ Indicating if new problem and functions are given or default problem is used
 - ◇ Selecting the initial memory size used by processes
 - ◇ Determining the period in which center should be updated on finished tasks



5- Developing examples

- » Complexity comes from different memory requirement and runtime
 - ◇ since logic should be maintained by user
- » Hence with dummy memory and estimated elapsed time we can create as complicated examples for the farm as needed [phase 2]



Resources

- » SAfarm (dynamically fed farm code by Dr. Wagner)
- » adlb
- » mrlib

Repository

<https://github.com/The-CodeBreakerR/FarmGen>



THANKS!

Any questions?