

# Practical Introduction for Parallel Optimization

Makoto Nakajima

Federal Reserve Bank of Philadelphia

April 2009

# Some General Tips

- Make sure the **serial code works flawlessly** before parallelizing.
- Think about **brute-force parallelization** (running multiple codes with different parameter values) before using parallel programming.
- Exploit all the **compiler options** before parallelizing. But start with lower-level optimization option when parallelizing, to avoid conflict between parallelization and the compiler's optimization algorithm.
- **Output everything** and make sure values which are supposed to be shared are actually shared across nodes.
- Learn how to kill all the processes in all nodes.  
If your code screws up, **CLEAN UP YOUR MESS** from all nodes.

# The Set-Up

## The Problem

$$\min_{x \in \mathbb{R}^n} f(x)$$

- Two natural ways to use parallelization:
  - Parallelize  $f(x)$
  - Parallelize  $\min$  → focus of this presentation.

# Comparison

- Parallelize  $f(x)$ 
  - More complicated.
  - Once parallelized, can be used with any optimization algorithm.
  - Steady gain from parallel.
- Parallelize min
  - Less complicated. No need to touch inside  $f(x)$ .
  - Many ready-made codes available.
  - Gain from parallel could vary.

# Motivating Example [1/2]

- You want to "calibrate" the standard Aiyagari economy with:
  - Ad-hoc borrowing constraint  $\underline{b} < 0$
  - Labor-leisure decision (Cobb-Douglas between consumption and leisure)
  - Labor productivity shock follows AR(1)
- You want to calibrate:
  - Discount factor  $\beta$
  - Borrowing limit  $\underline{b}$
  - Persistence of earnings shock  $\rho_p$
  - Standard deviation of the earnings shock  $\sigma_p$
  - Cobb-Douglas parameter between consumption and leisure  $\eta$
- To match, simultaneously:
  - $\frac{K}{Y} = 2.75$
  - Proportion of borrowers = 10%
  - Average proportion of time spent on working = 33%
  - Earnings Gini = 0.4
  - Autocorrelation of earnings = 0.9

# Motivating Example [2/2]

- $x = (\beta, \underline{b}, \rho_p, \sigma_p, \eta)$
- Obviously,  $n = 5$  (Dimension of  $x$ )
- $t^* = (2.75, 0.1, 0.33, 0.4, 0.9)$
- $\tilde{t}(x)$  is generated by the model, given the steady state of the model with a set of parameters  $x$ .
- $w$  is a vector of weights attached to each of the targets.  
(assume it is fixed)
- $f(x) = \sum_{i=1}^n w_i (t_i^* - \tilde{t}_i(x))^2$

# Method 1: Simplex (Nelder-Mead) Method

- Basic idea of the simplex method (details omitted):
  - 1 Start from  $n + 1$  points:  $x_0, x_1, \dots, x_n$
  - 2 Evaluate  $n + 1$  points and obtain  $f_0, f_1, \dots, f_n$
  - 3 Pick the point with the largest (worst)  $f$ . Assume it is  $x_n$ .
  - 4 Replace  $x_n$  by  $\tilde{x}_n$ .
- Naturally serial algorithm (not parallel)!
- Lee and Wiswall (2007):
  - Suppose we use  $m < n$  nodes.
  - Replace the worst  $m$  points simultaneously in Step 3 and 4.
- Gain from parallelization constrained by  $n > m$ .

## Method 2: Genetic Algorithm (GA)

- Basic idea of the very standard GA:
  - 1 Population of size  $N$ :  $x_1, \dots, x_N$
  - 2 Each  $x_i$  carries  $n$  chromosomes.
  - 3 Pick  $m$  pairs of parents. Assign a higher probability for  $x_i$  with a smaller error, so that the fit have better chances of mating.
  - 4 From each pair of parents, create two offsprings. Could use:
    - Crossover (mix chromosomes of parents)
    - Mutation (Could be completely random draw of  $x$ )
  - 5 Replace the  $2m$   $x$ 's with largest errors by  $2m$  offsprings.
- Naturally parallel!  
(Procedure for  $m$  pairs of parents are independent of others).
- Hybrid (with deterministic optimization algorithm) is also used.

# References

**Lee, Donghoon and Matthew Wiswall**, "A Parallel Implementation of the Simplex Function Minimization Routine," *Computational Economics*, 2007, 30 (2), 171–187.