UNIL | Université de Lausanne

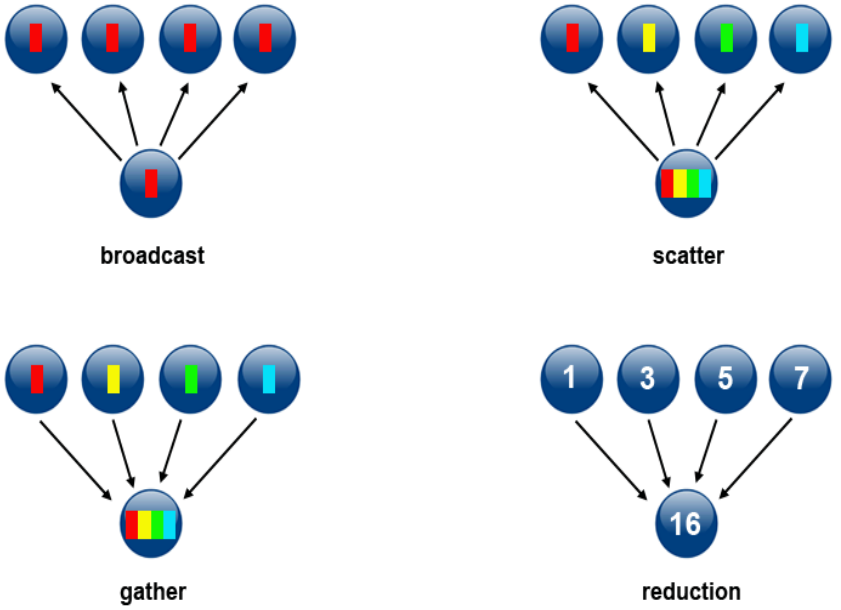
Exercise sheet – MPI

Simon Scheidegger
simon.scheidegger@gmail.com
July 23th, 2019
Open Source Economics Laboratory – BFI/UChicago

Supplementary material for the exercises is provided in
OSM2019/day3/Exercise_day3/supplementary_material_mpi

Including adapted teaching material from of G. Hager & G. Wellein,
B. Barney and the Swiss Supercomputing Centre (CSCS)

Reminder – collective communication



1. MPI exercise – Bcast

- Have a look at the code:
  OSE2019/day3/supplementary_material_mpi/broadcast.f90 or broadcast.cpp

- write a makefile to compile the code.

- hard code a value (any double precision value), and broadcast the value of rank 0 to all Ranks.

- submit the job via slurm.

2. MPI exercise – Allreduce

- Have a look at the code:
  OSE2019/day3/supplementary_material_mpi/allreduce.f90
  or allreduce.cpp

- write a makefile to compile the code.

- calculate the sum of all ranks.

- submit the job via slurm.

3. MPI exercise – Scatter

- Have a look at the code:
  OSE2019/day3/supplementary_material_mpi/scatter.f90
  or scatter.cpp

- write a makefile to compile the code.

- scatter the value of senddata of rank 0 to receivedata of all ranks.

- submit the job via slurm.

4. MPI Exercise (I)

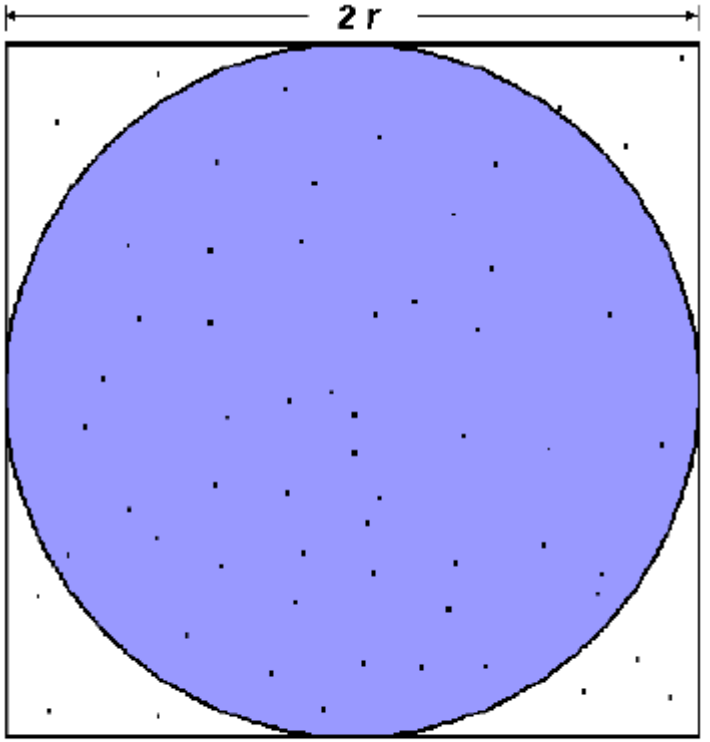
Revisit the following method of approximating PI:

1. Inscribe a circle in a square.

2. Randomly generate points in the square.

3. Determine the number of points in the square that are also in the circle.

4. approximate PI.



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

MPI exercise (II)

Serial pseudo code for this procedure:

```
npoints = 10000
circle_count = 0

do j = 1,npoints
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

PI = 4.0*circle_count/npoints
```

→ "embarrassingly parallel" solution:

- Break the loop iterations into chunks that can be executed by different tasks simultaneously.
- Each task executes its portion of the loop a number of times.
- Each task can do its work without requiring any information from the other tasks (there are no data dependencies).
- Master task receives results from other tasks using send/receive point-to-point operations.

```
npoints = 10000
circle_count = 0

p = number of tasks
num = npoints/p

find out if I am MASTER or WORKER

do j = 1,num
  generate 2 random numbers between 0 and 1
  xcoordinate = random1
  ycoordinate = random2
  if (xcoordinate, ycoordinate) inside circle
  then circle_count = circle_count + 1
end do

if I am MASTER
  receive from WORKERS their circle_counts
  compute PI (use MASTER and WORKER calculations)
else if I am WORKER
  send to MASTER circle_count
endif
```

→ **red: highlights changes for parallelism.**

→ **Try to implement this example (probably using a reduction clause).**

5. Work on Projects: American Options

- Parallelize the problem with MPI (for both Europen as well as the Asian option) (look at BS.cpp – take care of the random seeds!).
- **Check the speed-up** for combinations of threads and a variety of **sample points** on one node of MIDWAY (use slurm and the MPI_Wtime()).
- Generate Speed-up graphs (normalize to the 1 CPU result).
- Ensure that the serial and parallel return the same results.

5. Pricing an Asian Option

The following algorithm illustrates the steps in simulating n paths of m transitions each. To be explicit, we use Z_{ij} to denote the j-th draw from the normal distribution along the i-th path:

$$
\begin{aligned}
&\text{for } i = 1, \ldots, n \\
&\quad \text{for } j = 1, \ldots, m \\
&\quad\quad \text{generate } Z_{ij} \\
&\quad\quad \text{set } S_i(t_j) = S_i(t_{j-1}) \exp\left([r - \tfrac{1}{2}\sigma^2](t_j - t_{j-1}) + \sigma\sqrt{(t_j - t_{j-1})} Z_{ij} \right) \\
&\quad \text{set } \bar{S} = (S_i(t_1) + \cdots + S_i(t_m))/m \\
&\quad \text{set } C_i = e^{-rT}(\bar{S} - K)^+ \\
&\text{set } \hat{C}_n = (C_1 + \cdots + C_n)/n
\end{aligned}
$$

6. Project: Discrete State DP

- Parallelize the problem with MPI (look at the routine solver.cpp).
- **Check the speed-up** for combinations of threads and a variety of **the discretization level** on one node of **MIDWAY** (use slurm).
- Generate Speed up graphs (normalize to the 1 CPU result).
- Ensure that the serial and parallel return the same results.
- BONUS* task – Split the communicator (1 group per shock)

*we will quickly look at this in class on August 3rd

6. Recall DSDP

$$V_{new}(k,\Theta) = \max_{c} \left(u(c) + \beta \mathbb{E}\{V_{old}(k_{next},\Theta_{next})\}\right)$$

$$\text{s.t.} \quad k_{next} = f(k,\Theta_{next}) - c$$

$$\Theta_{next} = g(\Theta)$$

States of the model:

- k : today's capital stock → **There are many independent k's**
- Θ: today's productivity state → **The Θ's are independent**

Choices of the model:

- k_{next}

→ k, k_{next} , Θ and Θ_{next} are limited to a finite number of values

6. solver.cpp

```
for (int itheta=0; itheta<ntheta; itheta++) {

    /*
     Given the theta state, we now determine the new values and optimal policies corresponding to each
     capital state.
     */

    for (int ik=0; ik<nk; ik++) {

        // Compute the consumption quantities implied by each policy choice
        c=f(kgrid(ik), thetagrid(itheta))-kgrid;

        // Compute the list of values implied implied by each policy choice
        temp=util(c) + beta*ValOld*p(thetagrid(itheta));

        /* Take the max of temp and store its location.
         The max is the new value corresponding to (ik, itheta).
         The location corresponds to the index of the optimal policy choice in kgrid.
         */
        ValNew(ik, itheta)=temp.maxCoeff(&maxIndex);

        Policy(ik, itheta)=kgrid(maxIndex);

    }
}
```

2). *split MPI communicator

1). distribute k's via MPI

loops to worry about

7. Project: Continuous State DP

- Parallelize the non-stochastic Growth model, implemented with TASMANIAN using MPI (distribute the sparse grid points equally among MPI processes).
- **Check the speed-up** for combinations of threads and a variety of **the discretization level** on one node of **MIDWAY** (use slurm).
- Generate Speed up graphs (normalize to the 1 CPU result).
- Ensure that the serial and parallel return the same results.
- BONUS task: parallelize the stochastic growth model.
  In particular, split the communicator (1 group per shock)

7. Recall VFI: there are many **k's**

$$V(\mathbf{k}) = \max_{\mathbf{I},\mathbf{c},\mathbf{l}} \left(u(c,l) + \beta \left\{ V_{next}(k^+) \right\} \right),$$

$$s.t.$$

$$k_j^+ = (1-\delta)\cdot k_j + I_j \quad , \quad j = 1, ..., D$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left(\frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, ..., D$$

$$\sum_{j=1}^{D} (c_j + I_j - \delta \cdot k_j) = \sum_{j=1}^{D} (f(k_j, l_j) - \Gamma_j)$$

State k: sparse grid coordinates → **independent optimization problems**

V_{next} : sparse grid interpolator from the previous iteration step

Solve this optimization problem at every point in the sparse grid!

Attention: Take care of the econ domain/ sparse grid domain

7. Growth model – Homework (II)

→ Model with stochastic production

$$f(k_i, l_i, \theta_i) = \theta_i A k_i^{\psi} l_i^{1-\psi}$$

→ Here we assume 5 possible values of
Θ_i = {0.9, 0.95, 1.00, 1.05, 1.10}

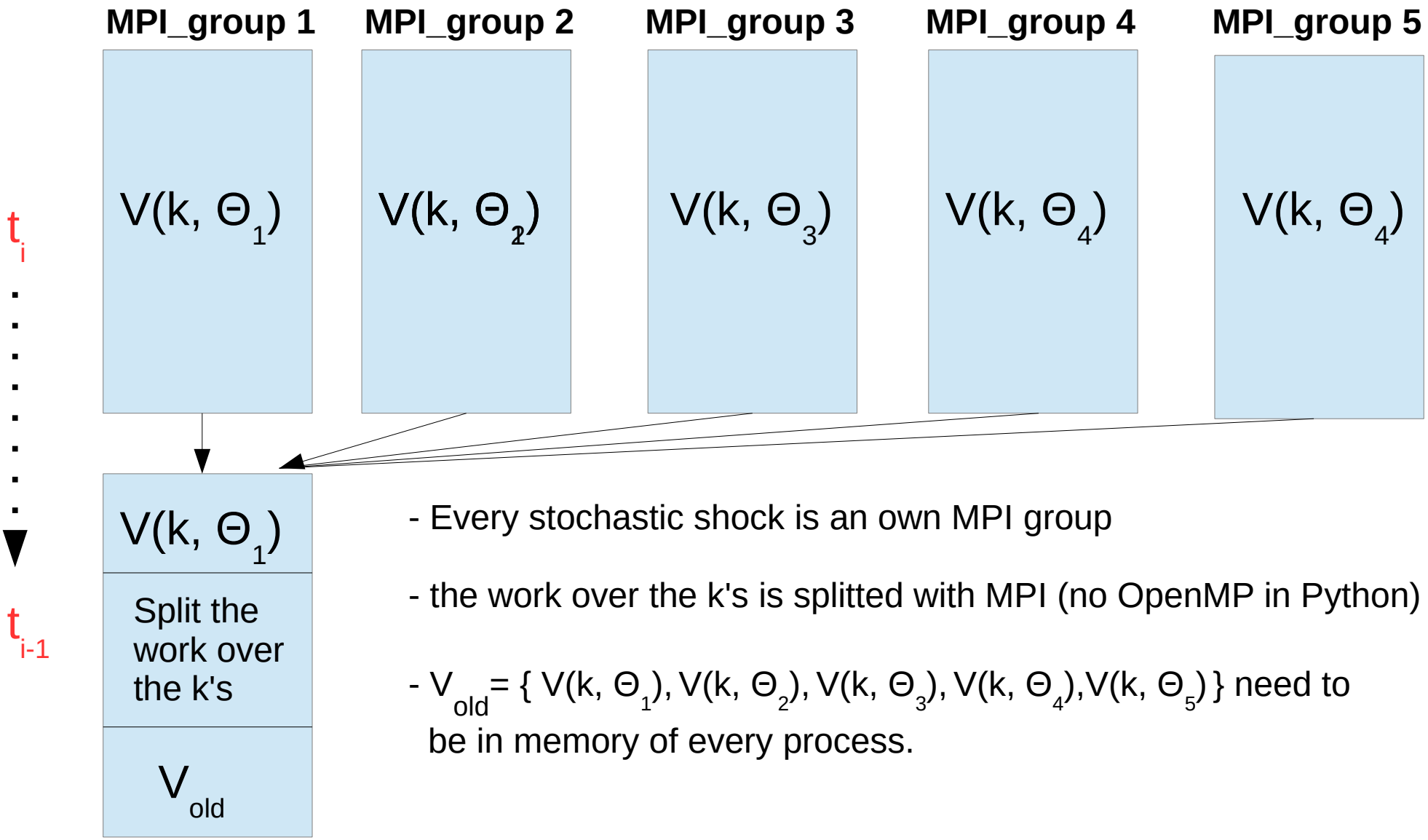
→ for simplicity, we assume **Π(*,*) = 1/5** (no Markov chain)

→ solve

$$V_t(k, \theta) = \max_{c,l,I} u(c, l) + \beta \mathbb{E}\left\{ V_{t+1}(k^+, \theta^+) \mid \theta \right\}$$

→ **Try to parallelize over the Θ's as well, they are independent**

The parallelization scheme



- Every stochastic shock is an own MPI group

- the work over the k's is splitted with MPI (no OpenMP in Python)

- $V_{old} = \{ V(k, \Theta_1), V(k, \Theta_2), V(k, \Theta_3), V(k, \Theta_4), V(k, \Theta_5) \}$ need to be in memory of every process.