HOOVER INSTITUTION
STANFORD UNIVERSITY

Universität Zürich UZH

# Hybrid parallel programming and DP – hands-on examples

Simon Scheidegger
simon.scheidegger@gmail.com

August 3rd, 2017
Open Source Macroeconomics Laboratory – BFI/UChicago

Including adapted teaching material from books, lectures and presentations by
B. Barney, B. Cumming, W. Gropp, G. Hager, M. Martinasso, R. Rabenseifner, O. Schenk, G. Wellein

# Outline

**I. Continuous state dynamic programming (growth model)**

→ MPI groups over stochastic states (not “hybrid” yet).

**II. Discrete state dynamic programming**

→ OMP & MPI

→ MPI groups

# 1. Continuous state DP – Growth model

→ Stochastic production to the model

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

→ Here we assume 5 possible values of
**$\Theta_i$ = {0.9, 0.95, 1.00, 1.05, 1.10}**

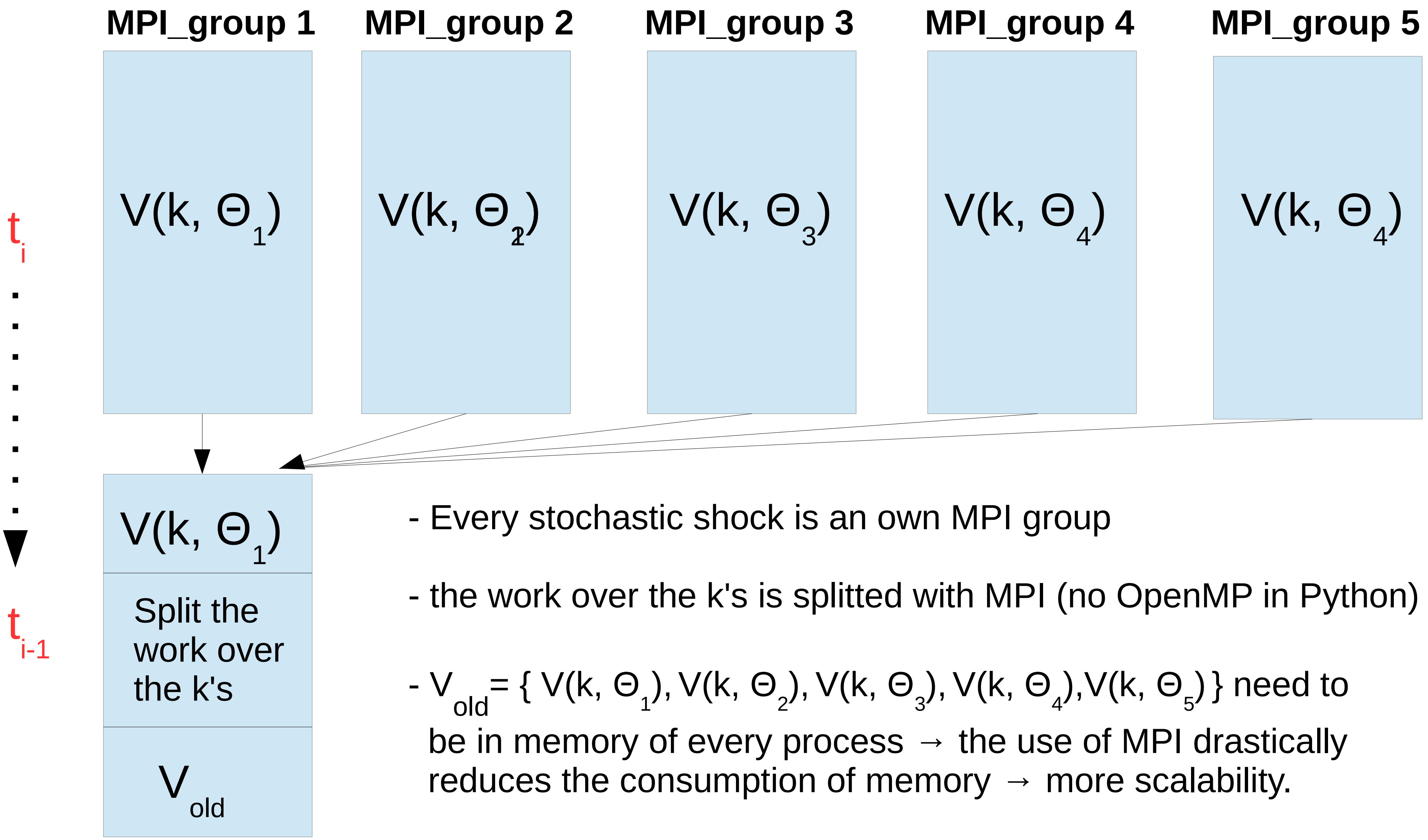
→ for simplicity, we assume **Π(*,*) = 1/5** (no Markov chain)

→ 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 → the use of MPI drastically reduces the consumption of memory → more scalability.

# Let's look at the code and run it

**cd /project2/osmlab/growth_model_multicomm**

```
>vi main.py
```

```
>vi interpolation.py
```

**Run**

>run on Midway (submit_python_midway.sh)

# DSDP – the model

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

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

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

States of the model:

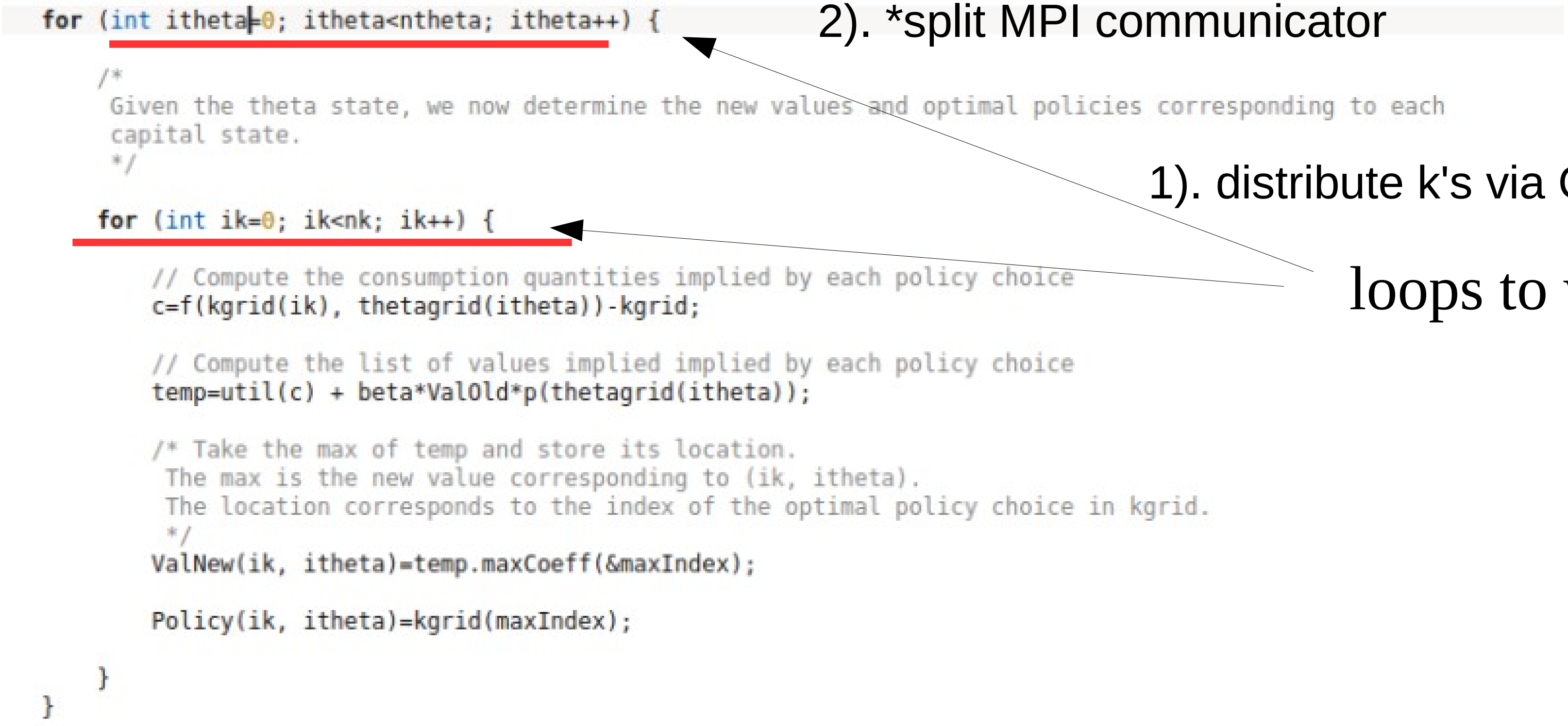
- k : today's capital stock → **There are many independent k's**
- Θ: today's productivity state → **The Θ's are independent**

Choices of the model:

- $k_{next}$

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

# solver.cpp – the critical loops



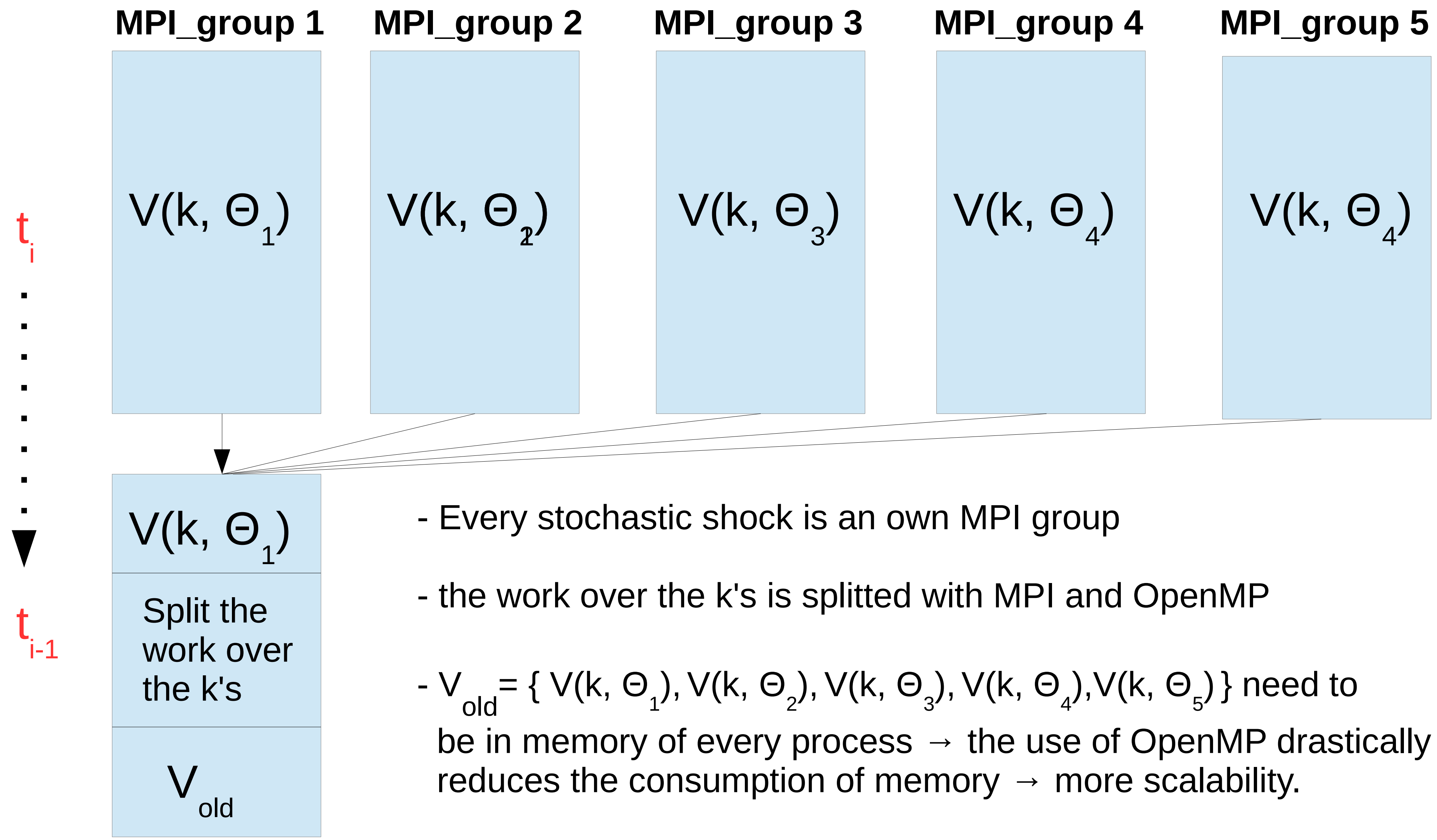
```
for (int itheta=0; itheta<ntheta; itheta++) {

    /*
     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 OpenMP & MPI

loops to worry about

# The parallelization scheme



- Every stochastic shock is an own MPI group

- the work over the k's is splitted with MPI and OpenMP

- $V_{old}$= { $V(k, \Theta_1)$, $V(k, \Theta_2)$, $V(k, \Theta_3)$, $V(k, \Theta_4)$,$V(k, \Theta_5)$ } need to be in memory of every process → the use of OpenMP drastically reduces the consumption of memory → more scalability.

# Let's look at the code and run it

**cd /project2/osmlab/DP_MultComms**

>vi main.cpp

>vi solver.cpp

**Compile**

>make (notice the compilation flags for OpenMP)

>run on Midway (submit_hybrid_midway.sh)