Programming FPGAs for Economics: An Introduction to Electrical Engineering Economics

Replication Package

README

Bhagath Cheela* André DeHon†

Jesús Fernández-Villaverde‡ Alessandro Peri§

September 25, 2024

*Department of Electrical and Systems Engineering, University of Pennsylvania, cheelabhagath@gmail.com

†Department of Electrical and Systems Engineering, University of Pennsylvania, andre@acm.org

‡Department of Economics, University of Pennsylvania, jesusfv@econ.upenn.edu

§Department of Economics, University of Colorado, Boulder, alessandro.peri@colorado.edu

Contents

1 Overview

This document serves as the `README` file for replicating the results presented in the paper "*Programming FPGAs for Economics: An Introduction to Electrical Engineering Economic*", by Bhagath Cheela, André DeHon, Jesús Fernández-Villaverde and Alessandro Peri.

This document focuses on the steps required for replication. Interested readers are invited to explore our comprehensive tutorial Cheela et al. (2023) for a thorough examination of the content within each file, and our `Github` repository for additional material (https://github.com/AleP83/FPGA-Econ.git). The replication code is organized in two steps:

- *Analysis*: estimation of the Krusell and Smith (1998) model across alternative software and hardware devices (Section 4);
- *Post-analysis*: collect and process the results from the *Analysis* step to populate *every* table and floating number in the body and online appendix of the paper (Section 5).

This work utilizes Amazon Web Services (AWS), with an estimated replication cost of $300 (based on 2024 AWS pricing) and approximately one week of supervised computational time.

2 Computational requirements

The *Analysis* is performed using the cloud services provided by Amazon Web Services (AWS) as of June 6th, 2024. Instructions for creating an AWS account can be found here. For learning how to launch an AWS instances follow this link. The software environment used to initialize each instance varies across exercises and is documented in the supplementary material located in `./documents`: CPU-compile.pdf, CPU-run.pdf, FPGA-design.pdf, and FPGA-run.pdf. These documents provide graphical walkthroughs for launching and initializing the instances, as well as executing the exercises. Section 4 directs the users to the relevant files as needed. The hardware specifications of the instances utilized are detailed in Online Appendix B.

The *Post-analysis* is conducted on an Apple M1 Max MacBook Pro with 64Gb of Memory (henceforth, referred to `Personal Desktop`) using Matlab R2022a Update 3 (9.12.0.1975300) (henceforth, referred to as `Matlab`). The replication requires less than 1GB of storage in the `Personal Desktop`.

3 Directory Structure

The replication directory is structured as follows.

```
./code
./documents
./I_estimation_results
./II_post_analysis
./III_floats
./IV_paper
```

Note: We recommend organizing the folders alphabetically to facilitate a clear visualization of the operational flow.
In particular:

- `./code`: contains the code used to estimate the Krusell and Smith (1998) model;
- `./documents`: contains our tutorial Cheela et al. (2023) and supporting documents with essential information for replicating our FPGA results;
- `./I_estimation_results`: here, you can access the output generated from model estimation across alternative software-devices;
- `./II_post_analysis`: contains the script main.m that generates all tables and floats for the paper. Results are stored in `./III_floats` and `./IV_paper/results`;
- `./III_floats`: this folder contains the binary files with all tables in the paper which require model estimates;
- `./IV_paper/results`: houses the `.txt` files created during the post-analysis, which are later used to the populate numerical data in the paper via the LaTeX file parser.tex.

4 Computing the Model across Software-Devices

This section carries out the *Analysis* step, estimating the Krusell and Smith (1998) model across various software and hardware configurations. The output of this analysis is stored in the `./I_estimation_results` directory, organized into appropriately named subfolders. In the *Post-analysis* step (Section 5), this output is processed to generate the numbers reported in the manuscript.

Since our work involves collecting time performance outcomes that inherently vary slightly between executions due to uncontrollable physical factors (e.g., circuit heat) and user-independent conditions (e.g., lack of seed control), we have pre-populated the subfolders in `./I_estimation_results` with results from our own executions to facilitate replication. For those wishing to replicate the entire process from scratch, we also provide ./replication-package-empty-folders.tar.gz, containing the folder structure without our results.

This section is organized modularly, allowing replicators to skip software-device exercises they find less relevant. To save time, `./I_estimation_results`/s3-bucket-2024-05-05.tar.gz includes pre-built executables and results for the computationally intensive CPU-C (Subsection 4.3) and FPGA (Subsection 4.4) exercises.
Unless otherwise specified, the time required for each task in this section is negligible (less than 10 minutes), provided the hardware specified for that task.

4.1 GPU

Folder: `./code`/gpu
Description: To generate the model estimates on the `GPU` using `Numba Cuda compiler` on an `NVIDIA GPU` (A100, in our work) follows this step:

- **Initialization.** Copy the agshock.txt and idshock.txt files from the directory `./code/common/shocks/` and paste them into `./code/gpu/input`
- **Execution.** Run on the terminal: `python3 ks_gpu.py > output`
- **Output.** Move the output file to `./I_estimation_results/gpu`

Remark: This analysis requires access to an `NVIDIA GPU`.
Hardware: This is the only step of the *Analysis* not conducted on AWS instances. Instead, it utilized an AMD EPYC 'Milan' CPU paired with an Nvidia A100 GPU, accessed via the CU Boulder Alpine Supercomputer (link).

4.2 CPU-Matlab

Folder. `./code/matlab/`.
Description. To generate the model estimates on the `CPU` using `Matlab` run: master.m. This script calls the function MMV_func.m, which modifies the original Maliar et al. (2010)'s script MAIN.m, to automatically produce model estimates for alternative grid sizes.
Output. The resulting model estimates are saved in 6 binary files located in the folder `./I_estimation_results/matlab/MMV/`.
Remark. Differently from the original Maliar et al. (2010)'s code, we replaced the spline interpolation with the linear interpolation. In addition, to guarantee replication across different software and devices, we need to fix aggregate and idiosyncratic shocks. To do so: *(i)* we set the Mersenne Twister random number generator's seed to 1 in line 9 of `./code/matlab/SHOCK.m` to fix aggregate and idiosyncratic shocks; *(ii)* we add lines 103-104 MMV_func.m to store the shocks in the folder `./code/common/shocks/`. *Note*: these lines are commented out as we need to generate and store the shocks only once.
Hardware: The replicator can reproduce these results on their `Personal Desktop`. For comparability, to collect timing performance data, we utilized the AWS m5n.large instance. For instructions on how to run Matlab on AWS, refer to this guide.

4.3 CPU-C

Folders: `./code/common` and `./code/fpga`
Description. To implement the Krusell and Smith (1998)'s algorithm on the `CPU`:

1. Compile all the binaries.
2. Execute the binaries on AWS.

4.3.1 Compile all the binaries

1. **Launch the Instance**. Log into the AWS instance m5n.large. To set up and launch the instance, follow the instructions in documents/CPU-run.pdf.
2. **Install the Packages**. Initiate a terminal session on the AWS instance and run the subsequent script to install the utilities `git, make, tmux and wget`:

```
sudo yum install git -y
sudo yum install make -y
sudo yum install tmux -y
sudo yum install wget -y
```

3. **Clone the GitHub repositories**. Clone our GitHub repository into a directory of your preference (e.g., /home/ec2-user):

```
git clone https://github.com/AleP83/FPGA-Econ.git
```

4. **Set the AWS credentials.** Configure your AWS credentials by executing the following command in the terminal:

```
aws configure
```

Follow the steps here:

```
$ aws configure
AWS Access Key ID [**************xxxx]: <Your AWS Access Key ID>
AWS Secret Access Key [**************xxxx]: <Your AWS Secret Access
   Key>
Default region name: us-west-2
Default output format [None]: json
```

For more information visit this link.

5. **Install** `OpenMPI`. Run the following script from the terminal:

```
sh code/common/util/OpenMPI_install.sh
```

Note: Installing Open-MPI may take some time (10-15 minutes).

6. **Set the OpenMPI environment**. If you are compiling or building for parallel execution, execute the following commands in the terminal from the parent directory:

```
export PATH=$PATH:$HOME/openmpi/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/openmpi/lib
```

7. **Modify the Makefile.** Update settings in the code/Makefile as follows:

   - **Set the AWS S3 Bucket Name:** Specify the S3 bucket name by replacing S3-NAME-GOES-HERE

   ```
   S3_EXE_BUCKET_NAME := S3-NAME-GOES-HERE
   ```

   Remark: The S3 bucket name must be globally unique within AWS. If an error occurs during bucket creation, it may be due to the name being already in use by another user.

   - **Select the AWS region** of the S3 bucket (default is us-west-2):

   ```
   AWS_REGION := us-west-2
   ```

8. **Modify the Main.** Open /code/common/app.cpp and set the number of models N_MODEL you want to compute (1,200 in our benchmark specification):

```
#define N_MODEL 1200    // total number of models
```

9. **Set the Grid Sizes.** Open /code/common/definitions.h and set the grid sizes:

```
#define NKGRID 100       // grid points on individual capital grid
#define NKM_GRID 4       // grid points on aggregate capital grid
```

The benchmark code is set to allocate `NKGRID=100`, `NKM_GRID=4`.

10. **Set the Software Design.** Open /code/common/dev_options.h and select the interpolation-range search algorithm:

```
// Set only one of the following macros to 1, keeping the rest to
   zero.
#define _LINEAR_SEARCH 0
#define _BINARY_SEARCH 0
#define _CUSTOM_BINARY_SEARCH 1
```

The benchmark code is set to implement the jump-search algorithm _CUSTOM_BINARY_SEARCH 1.

11. **Compile the binary**. After modifying the files, navigate to the /code directory using the terminal. Then, compile the application for CPU execution using the following command:

- For building binaries for sequential execution on single-core instance:

```
make cpu_to_s3 CPU_EXE=<# Economies>_<#indiv cap.>_<#agg cap.>
```

For example, compile the benchmark model as follows:

```
make cpu_to_s3 CPU_EXE=1200_100k_4km
```

- For building binaries for parallel execution on multi-core instance:

```
make openmpi_to_s3 OPENMPI_EXE=mpi_<# Economies>_<# indiv capital
   >_<# agg capital>
```

For example, compile the benchmark model as follows:

```
make openmpi_to_s3 OPENMPI_EXE=mpi_1200_100k_4km
```

12. **Compile all binaries.** To streamline the process, Table 1-9 in Appendix A concisely summarizes all manual changes to the code required to compile binaries for all of the combinations, `NKGRID`$\in \{100, 200, 300\}$, `NKM_GRID`$\in \{4, 8\}$, and search algorithms $\in \{linear, binary, custom\_binary\}$, required to replicate the results in the paper.

```
make cpu_to_s3 CPU_EXE=1200_100k_4km
make cpu_to_s3 CPU_EXE=1200_200k_4km
make cpu_to_s3 CPU_EXE=1200_300k_4km
make cpu_to_s3 CPU_EXE=1200_100k_8km
make cpu_to_s3 CPU_EXE=1200_200k_8km
make cpu_to_s3 CPU_EXE=1200_300k_8km
make cpu_to_s3 CPU_EXE=1200_linear
make cpu_to_s3 CPU_EXE=1200_binary
make openmpi_to_s3 OPENMPI_EXE=mpi_1200_100k_4km
```

Output: The make cpu_to_s3 and make openmpi_to_s3 commands will save the binaries in your S3 bucket, identified as `$S3_EXE_BUCKET_NAME`, under the folder s3://$S3_EXE_BUCKET_NAME/executables/cpu/:

```
$S3_EXE_BUCKET_NAME/
    executables/
        cpu/
            1200_100k_4km
            1200_200k_4km
            1200_300k_4km
            1200_100k_8km
            1200_200k_8km
            1200_300k_8km
            1200_linear
            1200_binary
            mpi_1200_100k_4km
```

4.3.2 Execute the binaries on AWS

1. **Launch the Instance**. Log into the appropriate AWS instance: m5n.large, m5n.4xlarge, or m5n.24xlarge. To set up and launch the instance, follow the instructions in documents/CPU-run.pdf.

2. **Install the Packages**. Initiate a terminal session on the AWS instance and run the subsequent script to install the utilities `git, make, tmux and wget`:

```
sudo yum install git -y
sudo yum install make -y
sudo yum install tmux -y
sudo yum install wget -y
```

3. **Clone the GitHub repositories.** Clone our GitHub repository into a directory of your preference (e.g. /home/ec2-user):

```
git clone https://github.com/AleP83/FPGA-Econ.git
```

4. **Set the AWS credentials**. Configure your AWS credentials by executing the following command in the terminal:

```
aws configure
```

Follow the steps here:

```
$ aws configure
AWS Access Key ID [**************xxxx]: <Your AWS Access Key ID>
AWS Secret Access Key [**************xxxx]: <Your AWS Secret Access
   Key>
Default region name: us-west-2
Default output format [None]: json
```

For more information visit this link.

5. **Modify the Makefile.** Update settings in the code/Makefile as follows:

   - **Set the AWS S3 Bucket Name:** Specify the S3 bucket name by replacing S3-NAME-GOES-HERE

   ```
   S3_EXE_BUCKET_NAME := S3-NAME-GOES-HERE
   ```

   Remark: The S3 bucket name must be globally unique within AWS. If an error occurs during bucket creation, it may be due to the name being already in use by another user.

   - **Select the AWS region** of the S3 bucket (default is us-west-2):

   ```
   AWS_REGION := us-west-2
   ```

6. **Modify Shell Script for CPU Results**. Update settings in the code/common/util/generate_cpu_results.sh as follows:

   - **Set the AWS S3 Bucket Name:** Specify the S3 bucket name by replacing S3-NAME-GOES-HERE

   ```
   S3_EXE_BUCKET_NAME="S3-NAME-GOES-HERE"
   ```

   Remark: The S3 bucket name must be globally unique within AWS. If an error occurs during bucket creation, it may be due to the name being already in use by another user.

   - **Select the AWS region** of the S3 bucket (default is us-west-2):

   ```
   AWS_REGION="us-west-2"
   ```

   AWS Region and Bucket name should coincide with the ones used in the compiling stage.

7. **Initiate `tmux` terminal session.** To ensure your terminal session remains active throughout the potentially lengthy execution, initiate a terminal multiplexer session:

   ```
   tmux
   ```

   The `tmux` command allows you to detach and reattach to terminal sessions without interruption. For example, to resume a `tmux` session with index 0, use the following command:

```
tmux attach -t 0
```

For detailed instructions on how to use `tmux`, see this guide.

8. **Run all binaries.** To run the binaries on the CPU, navigate to the directory /code from within the `tmux` terminal window. Therein, execute the binaries sequentially and copy the generated results to AWS-S3 Bucket with the following AWS instance specific commands:

   - To replicate results on the m5n.large instance execute on the `tmux` terminal:

     ```
     make cpu_results M5N=1x USE_AWS_S3_EXE=yes
     ```

     Execution time: This step takes about one week (approximately six and a half days). To expedite the process (down to 50 hours), we provide commands to split the workload into three batches. These batches can be executed concurrently on three distinct m5n.large instances. This is achieved by replacing M5N=1x in the command with M5N=1xBATCH1, M5N=1xBATCH2, and M5N=1xBATCH3. For instance, to initiate the first batch, the following command can be used:

     ```
     make cpu_results M5N=1xBATCH1 USE_AWS_S3_EXE=yes
     ```

   - To replicate results on the m5n.4xlarge instance execute on the `tmux` terminal:

     ```
     make cpu_results M5N=4x USE_AWS_S3_EXE=yes
     ```

     Execution time: About one hour.

   - To replicate results on the m5n.24xlarge instance execute on the `tmux` terminal:

     ```
     make cpu_results M5N=24x USE_AWS_S3_EXE=yes
     ```

     Execution time: About 10 minutes.

Output. The command make cpu_results automatically saves the results in your S3 bucket, identified as `$S3_EXE_BUCKET_NAME`, under the folder s3://$S3_EXE_BUCKET_NAME/results/cpu/:

```
$S3_EXE_BUCKET_NAME/
    results/
        cpu/
            *.txt
```

4.4 FPGA

Folders: ./code/common and ./code/fpga
Description. To implement the Krusell and Smith (1998) algorithm on the FPGA:

1. Synthesize the application in hardware.
2. Execute the application on an FPGA instance.

4.4.1 Synthesize the application in hardware

The replication package provides the pre-synthesized images in the directory ./code/executables/fpga/fpga_afi and associated host binaries in ./code/executables/fpga/host_executables. To create these images follow these steps:

1. **Launch the Instance**. Log into the AWS build instance: z1d.2xlarge. To set up and launch the instance, follow the instructions in documents/FPGA-design.pdf.
2. **Clone the GitHub repositories**. Open the terminal. Then, clone the AWS repository and our GitHub repository into a directory of your preference (e.g., /home/centos/):

```
git clone https://github.com/aws/aws-fpga.git $AWS_FPGA_REPO_DIR
git clone https://github.com/AleP83/FPGA-Econ.git
```

3. **Set the AWS credentials**. Configure your AWS credentials by executing the following command in the terminal:

```
aws configure
```

   Follow the steps here:

```
$ aws configure
AWS Access Key ID [**************xxxx]: <Your AWS Access Key ID>
AWS Secret Access Key [**************xxxx]: <Your AWS Secret Access
   Key>
Default region name: us-west-2
Default output format [None]: json
```

   For more information visit this link.

4. **Modify the Makefile.** Update settings in the code/Makefile as follows:

   - **Set the AWS S3 Bucket Name:** Specify the S3 bucket name by replacing S3-NAME-GOES-HERE

```
S3_EXE_BUCKET_NAME := S3-NAME-GOES-HERE
```

     Remark: The S3 bucket name must be globally unique within AWS. If an error occurs during bucket creation, it may be due to the name being already in use by another user.

- **Select the AWS region** of the S3 bucket (default is us-west-2):

```
AWS_REGION := us-west-2
```

5. **Modify the Main.** Open /code/common/app.cpp and set the number of models N_MODEL you want to compute (1,200 in our benchmark specification):

```
#define N_MODEL 1200    // total number of models
```

6. **Set the Grid Sizes.** Open /code/common/definitions.h and set the grid size:

```
#define NKGRID 100        // grid points on individual capital grid
#define NKM_GRID 4        // grid points on aggregate capital grid
```

7. **Set the Hardware Design.** Open /code/common/dev_options.h and select the FPGA design:

```
#define _BASELINE 0  // Design with no HLS acceleration.
#define _PIPELINE 0  // Design with only PIPELINE acceleration
#define _WITHIN_ECONOMY 0  // Single-Kernel Design
#define _ACROSS_ECONOMY 1  // Three-kernel Design (Benchmark)
```

8. **Set the Hardware Design Specs.** Open /code/fpga/design.cfg and select the single vs three-kernel design by appropriately commenting out the code you do not need. For example, the listing below executes the three-kernel design by commenting out (using #) the one-kernel design:

```
# Enable either single kernel or three kernel
[connectivity]
###############single kernel start#################
# nk=runOnfpga:1:runOnfpga_1
################three kernel start#################
nk=runOnfpga:3:runOnfpga_1.runOnfpga_2.runOnfpga_3
slr=runOnfpga_1:SLR2
slr=runOnfpga_2:SLR1
slr=runOnfpga_3:SLR0
sp=runOnfpga_1.m_axi_gmem0:DDR[1]
sp=runOnfpga_2.m_axi_gmem0:DDR[0]
sp=runOnfpga_3.m_axi_gmem0:DDR[3]
```

9. **Create all FPGA images.** To ensure your terminal session remains active throughout the potentially lengthy synthesis process, initiate a terminal multiplexer session:

```
tmux
```

The `tmux` command allows you to detach and reattach to terminal sessions without interruption. For example, to resume a `tmux` session with index 0, use the following command:

```
tmux attach -t 0
```

For detailed instructions on how to use `tmux`, see this guide.

To initiate the synthesis of the FPGA circuit, navigate to the directory /code from within the `tmux` terminal window. Therein, execute the following instructions to generate the host and fpga target files on the build instance (z1d.2xlarge); and subsequently, upload the resulting executables to the AWS bucket:

```
make clean
unset XCL_EMULATION_MODE
//setup environment
source $AWS_FPGA_REPO_DIR/vitis_setup.sh
export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM)
export XCL_EMULATION_MODE=hw
//build the target
make afi FPGA_BIN=<fpga_bin> HOST_BIN=<host_bin>
#E.g. make afi FPGA_BIN=3ker_100k_4km HOST_BIN=1200_3ker_100k_4km
```

In particular, follow this fpga_bin-host_bin naming convention:

```
make afi FPGA_BIN=3ker_100k_4km HOST_BIN=1200_3ker_100k_4km
make afi FPGA_BIN=1ker_100k_4km HOST_BIN=1200_1ker_100k_4km
make afi FPGA_BIN=1ker_200k_4km HOST_BIN=1200_1ker_200k_4km
make afi FPGA_BIN=1ker_300k_4km HOST_BIN=1200_1ker_300k_4km
make afi FPGA_BIN=1ker_100k_8km HOST_BIN=1200_1ker_100k_8km
make afi FPGA_BIN=1ker_200k_8km HOST_BIN=1200_1ker_200k_8km
make afi FPGA_BIN=1ker_300k_8km HOST_BIN=1200_1ker_300k_8km
make afi FPGA_BIN=baseline_1ker_100k_4km HOST_BIN=120_1ker_100k_4km
make afi FPGA_BIN=pipeline_1ker_100k_4km HOST_BIN=120_1ker_100k_4km
```

Table 10-18 in Appendix B concisely summarize all manual changes to the code required to synthesize all nine FPGA images used in the paper.

Estimated Run Time. The estimated time to build the three-kernel design is 8 hours. Single kernel design take on average less than 4 hours each.

Output: The command make afi automatically saves FPGA images and host binaries in your S3 bucket, identified as `$S3_EXE_BUCKET_NAME`. This process organizes the files in the folder s3://$S3_EXE_BUCKET_NAME/executables/fpga/ as follows:

- ./fpga_afi/<fpga_bin>: stores the FPGA images
- ./host_executables/<host_bin>: stores the host binaries that call the FPGA images

```
$S3_EXE_BUCKET_NAME/
    executables/
        fpga/
            fpga_afi/
                1ker_100k_4km.awsxclbin
                1ker_100k_8km.awsxclbin
                1ker_200k_4km.awsxclbin
                1ker_200k_8km.awsxclbin
                1ker_300k_4km.awsxclbin
                1ker_300k_8km.awsxclbin
```

```
            3ker_100k_4km.awsxclbin
            baseline_1ker_100k_4km.awsxclbin
            pipeline_1ker_100k_4km.awsxclbin
        host_executables/
            120_1ker_100k_4km
            1200_1ker_100k_4km
            1200_1ker_100k_8km
            1200_1ker_200k_4km
            1200_1ker_200k_8km
            1200_1ker_300k_4km
            1200_1ker_300k_8km
            1200_3ker_100k_4km
```

Remark: Once you are done with the creation of the FPGA images, delete all `S3` buckets, except for the one you created, `$S3_EXE_BUCKET_NAME`. For more information on how to delete `S3` buckets, follow this link.

4.4.2 Execute on an AWS FPGA instance

1. **Launch the Instance**. Log into the appropriate AWS instance: f1.2xlarge, f1.4xlarge, or f1.16xlarge. To set up the instance, follow the instructions in documents/FPGA-run.pdf.

2. **Clone the GitHub repositories**. Open the terminal. Then, clone our GitHub repository into a directory of your preference (e.g., /home/centos/):

   ```
   git clone https://github.com/AleP83/FPGA-Econ.git
   ```

3. **Set the AWS credentials**. Configure your AWS credentials by executing the following command in the terminal:

   ```
   aws configure
   ```

   Follow the steps here:

   ```
   $ aws configure
   AWS Access Key ID [**************xxxx]: <Your AWS Access Key ID>
   AWS Secret Access Key [**************xxxx]: <Your AWS Secret Access
      Key>
   Default region name: us-west-2
   Default output format: json
   ```

   For more information visit this link.

4. **Modify the Makefile.** Update settings in the code/Makefile as follows:

   - **Set the AWS S3 Bucket Name:** Specify the S3 bucket name by replacing S3-NAME-GOES-HERE

     ```
     S3_EXE_BUCKET_NAME := S3-NAME-GOES-HERE
     ```

Remark: The S3 bucket name must be globally unique within AWS. If an error occurs during bucket creation, it may be due to the name being already in use by another user.

- **Select the AWS region** of the S3 bucket (default is us-west-2):

```
AWS_REGION := us-west-2
```

AWS Region and Bucket name should coincide with the ones used in the synthesis stage.

5. **Modify Shell Script for FPGA Results**. Update settings in the code/common/util/generate_fpga_results.sh as follows:

   - **Set the AWS S3 Bucket Name:** Specify the S3 bucket name by replacing S3-NAME-GOES-HERE

```
S3_EXE_BUCKET_NAME="S3-NAME-GOES-HERE"
```

   Remark: The S3 bucket name must be globally unique within AWS. If an error occurs during bucket creation, it may be due to the name being already in use by another user.

   - **Select the AWS region** of the S3 bucket (default is us-west-2):

```
AWS_REGION="us-west-2"
```

   AWS Region and Bucket name should coincide with the ones used in the synthesis stage.

6. **Initiate `tmux` terminal session.** To ensure your terminal session remains active throughout the execution, initiate a terminal multiplexer session:

```
tmux
```

   The `tmux` command allows you to detach and reattach to terminal sessions without interruption. For example, to resume a `tmux` session with index 0, use the following command:

```
tmux attach -t 0
```

   For detailed instructions on how to use `tmux`, see this guide.

7. **Execute application on F1 instances.** Navigate to the code folder, and run the following commands to:

   - Copy the executables from AWS S3 folder to the current AWS instance;
   - Execute all the relevant exercises
   - Transfer the generated results into the S3 folder.

   In particular:

(a) To replicate results on the f1.2xlarge instance execute on the `tmux` terminal:

```
make fpga_results TABLE=all USE_AWS_S3_EXE=yes
```

Estimated time: Approximately 20 hours.

(b) To replicate results on the f1.4xlarge and f1.16xlarge instance execute on the `tmux` terminal of the respective instance:

```
make fpga_results TABLE=3 USE_AWS_S3_EXE=yes
```

Estimated time: Approximately 10 minutes for both.

Output. The command make fpga_results automatically saves the results in your S3 bucket, identified as `$S3_EXE_BUCKET_NAME`, under the folder s3://$S3_EXE_BUCKET_NAME/results/fpga/:

```
$S3_EXE_BUCKET_NAME/
    results/
        fpga/
            *.txt
            *.csv
            *.run_summary
            *.rpt
            *.xtxt
            *.log
```

Remark: Make sure to terminate your F1 instance! Even the smaller one (f1.2xlarge) costs 1.65$/hr.

4.5 Transfer the results to your local folder

The S3 bucket named $S3_EXE_BUCKET_NAME contains the results of all CPU-C and FPGA-C model estimations. To download these results to your Personal Desktop, run the following file after making these changes:

- **Launch the Instance**. Log into an inexpensive AWS instance, say m5n.large.
- **Download S3 bucket in AWS instance**. Copy the S3 bucket into a directory of your choice within your AWS instance.

```
aws s3 cp --recursive s3://$S3_EXE_BUCKET_NAME/ ./s3-bucket/
```

- **Compress the results**. Compress the bucket results using tar

```
tar -czvf s3-bucket-$(date +%Y-%m-%d).tar.gz s3-bucket/
```

- **Copy the results in your local machine.** Navigate into your Personal Desktop to the directory I_estimation_results/ and execute the following commands

```
instance_name="35-91-136-136"
key_directory="<Your␣AWS␣Access␣Key␣ID>"
region="<Your␣region>"
scp -i "${key_directory}" ec2-user@ec2-$instance_name.$region.compute
   .amazonaws.com:/home/ec2-user/s3-bucket-*.tar.gz ./
```

  After downloading, unzip the file.

Software: Unix Shell.
Hardware: Personal Desktop with at least 1Gb of storage.

4.5.1 Clean AWS account

Once you are done with the AWS estimation, terminal all instances, delete all attached volumes and S3 buckets to avoid unintended charges.

5 Post Analysis

The *Post-analysis* is conducted on the `Personal Desktop`. The post analysis uses the script `./II_post_analysis/main.m` to process the inputs in `./I_estimation_results` to replicate tables and figures in the paper. The results of the post analysis are organized as follows:

- To organize the results for post analysis, navigate to the folder I_estimation_results, and execute the following commands to modify the file permissions of the script organize_files_for_post_analysis.sh and then execute it:

```
chmod u+x organize_files_for_post_analysis.sh
./organize_files_for_post_analysis.sh
```

- `./III_floats`: `./II_post_analysis/main.m` stores in the binary file `./III_floats`/-Tables.mat all tables referenced in the paper, which rely on data from the model estimation. Additionally, `./III_floats` is structured with subfolders, each dedicated to storing individual tables, as indicated by their respective names.

- `./IV_paper/results`: The script `./II_post_analysis/main.m` saves the results of the post-analysis in this folder, in the form of `.txt` files. These files are used by the LaTeX file `./IV_paper/results/parser.tex` to automatically populate numerical data of our paper.

For computing the Euler Equation Errors in Appendix Table A.5, the file `./II_post_analysis/main.m` calls the function `./II_post_analysis/EEErrors/EEE_fun.m`. This function computes the euler equation errors for policy functions estimated on the FPGA, CPU-C and CPU-Matlab. To compute the Euler equation errors we use the code in Maliar et al. (2010) (`Test.m`) adapted to receive different grid sizes in `./II_post_analysis/EEErrors/EE_MMV.m`.
Output: Results are stored in `./III_floats`, `./IV_paper/results`.
Software: `Matlab R2022a`, `Unix Shell`, LaTeX `TeXShop Version 5.42`.
Hardware: `Personal Desktop` with at least `1Gb` of storage.

5.1 List of Tables and Figures

This subsection provides a comprehensive list of tables and figures included in the paper. Tables generated by our code are underlined (e.g., Table X). Instructions for replicating figures and tables not produced by our code are also provided.

- **Table 1**: Calibrated Parameters based on Maliar et al. (2010).

- Table 2: Generated by `./II_post_analysis/main.m`, lines 460-509.

- Table 3: Panel A is generated by `./II_post_analysis/main.m`, lines 597-628. Panel B estimates implementation costs by comparing lines between the CPU and FPGA codebases. For the kernel segment, we count the `#pragmas` in `./code/fpga/hw.cpp`. For the non-kernel segment, we count the lines in `./code/common/app.cpp` for FPGA or OpenMPI-CPU execution. To facilitate the comparison, we report those in `./III_floats/table3/line-comparison/`.

- Table 4: Generated by `./II_post_analysis/main.m`, lines 630-752.

- Table 5: Generated by `./II_post_analysis/main.m`, lines 754-805.

- **Table A1**: List of abbreviations adopted in the paper.

- **Table A2**: Hardware architecture and AWS cloud pricing.
  Source: AWS instances, AWS specs.

- Table A3: Generated by `./II_post_analysis/main.m`, lines 1111-1132.

- Table A4: Generated by `./II_post_analysis/main.m`, lines 524-596.

- Table A5: Generated by `./II_post_analysis/main.m`, lines 716-740.

- Table A6: Generated by `./II_post_analysis/main.m`, lines 807-1109.

All figures were generated using `Microsoft PowerPoint v16.89.1`, except Figures 4 and 5, which were created using `Xilinx Vivado v2021.2 (64-bit)`. The placement analysis for these figures was performed following the steps outlined in this guide. For more details, refer to the Vivado Design Flows Overview Tutorial.

Data Availability and Provenance Statements

This paper does not involve analysis of external data; the only data are generated by the authors via simulation in their code.

Acknowledgements

The Matlab implementation utilizes source code from Maliar et al. (2010), available at https://lmaliar.ws.gc.cuny.edu/codes/, and distributed under the terms of use of the license therein.

References

Cheela, B., A. DeHon, J. Fernández-Villaverde, and A. Peri (2023). *A Beginner's Guide to Programming FPGAs for Economics: An Introduction to Electrical Engineering Economics*. University of Pennsylvania.

Krusell, P. and A. A. Smith (1998). Income and wealth heterogeneity in the macroeconomy. *Journal of Political Economy 106*(5), 867–896.

Maliar, L., S. Maliar, and F. Valli (2010). Solving the incomplete markets model with aggregate uncertainty using the Krusell-Smith algorithm. *Journal of Economic Dynamics and Control 34*(1), 42–49.

A Compile CPU Executables

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 100 #define NKM_GRID 4
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_100k_4km`

Table 1: $N_k = 100$, $N_M = 4$, 1200 Economies.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 200 #define NKM_GRID 4
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_200k_4km`

Table 2: $N_k = 200$, $N_M = 4$, 1200 Economies.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 300 #define NKM_GRID 4
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_300k_4km`

Table 3: $N_k = 300$, $N_M = 4$, 1200 Economies.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 100 #define NKM_GRID 8
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_100k_8km`

Table 4: $N_k = 100$, $N_M = 8$, 1200 Economies.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 200 #define NKM_GRID 8
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_200k_8km`

Table 5: $N_k = 200$, $N_M = 8$, 1200 Economies.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 300 #define NKM_GRID 8
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_300k_8km`

Table 6: $N_k = 300$, $N_M = 8$, 1200 Economies.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 100 #define NKM_GRID 4
`/common/dev_options.h`	#define _LINEAR_SEARCH 1 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 0
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_linear`

Table 7: $N_k = 100$, $N_M = 4$, 1200 Economies, Linear Search.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 100 #define NKM_GRID 4
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 1 #define _CUSTOM_BINARY_SEARCH 0
In the terminal	`make clean` `make cpu_to_s3 CPU_EXE=1200_binary`

Table 8: $N_k = 100$, $N_M = 4$, 1200 Economies, Binary Search.

File	Modify
`/common/app.cpp`	#define N_MODEL 1200
`/common/definitions.h`	#define NKGRID 100 #define NKM_GRID 4
`/common/dev_options.h`	#define _LINEAR_SEARCH 0 #define _BINARY_SEARCH 0 #define _CUSTOM_BINARY_SEARCH 1

In the terminal	`export PATH=$PATH:$HOME/openmpi/bin`
	`export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/openmpi/lib`
	`make clean`
	`make openmpi_to_s3 OPENMPI_EXE=mpi_1200_100k_4km`

Table 9: OpenMPI: $N_k = 100$, $N_M = 4$, 1200 Economies.

B Create the FPGA images

File	Modify
`/common/app.cpp`	`#define N_MODEL 1200`
`/common/definitions.h`	`#define NKGRID 100` `#define NKM_GRID 4`

`/common/dev_options.h`	`#define _BASELINE 0`
	`#define _PIPELINE 0`
	`#define _WITHIN_ECONOMY 0`
	`#define _ACROSS_ECONOMY 1`

`/fpga/design.cfg`	`### three kernel`
	`[connectivity]`
	`nk=runOnfpga:3:runOnfpga_1.runOnfpga_2.runOnfpga_3`

In the terminal	`tmux`
	`make clean`
	`unset XCL_EMULATION_MODE`
	`source $AWS_FPGA_REPO_DIR/vitis_setup.sh`
	`export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM)`
	`export XCL_EMULATION_MODE=hw`
	`make afi FPGA_BIN=3ker_100k_4km HOST_BIN=1200_3ker_100k_4km`

Table 10: Three-kernel design, $N_k = 100$, $N_M = 4$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 1200
/common/definitions.h	#define NKGRID 100 #define NKM_GRID 4
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 0 #define _WITHIN_ECONOMY 1 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=1ker_100k_4km HOST_BIN=1200_1ker_100k_4km

Table 11: Single-kernel design, $N_k = 100$, $N_M = 4$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 1200
/common/definitions.h	#define NKGRID 200 #define NKM_GRID 4
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 0 #define _WITHIN_ECONOMY 1 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=1ker_200k_4km HOST_BIN=1200_1ker_200k_4km

Table 12: Single-kernel design, $N_k = 200$, $N_M = 4$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 1200
/common/definitions.h	#define NKGRID 300 #define NKM_GRID 4
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 0 #define _WITHIN_ECONOMY 1 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=1ker_300k_4km HOST_BIN=1200_1ker_300k_4km

Table 13: Single-kernel design, $N_k = 300$, $N_M = 4$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 1200
/common/definitions.h	#define NKGRID 100 #define NKM_GRID 8
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 0 #define _WITHIN_ECONOMY 1 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=1ker_100k_8km HOST_BIN=1200_1ker_100k_8km

Table 14: Single-kernel design, $N_k = 100$, $N_M = 8$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 1200
/common/definitions.h	#define NKGRID 200 #define NKM_GRID 8
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 0 #define _WITHIN_ECONOMY 1 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=1ker_200k_8km HOST_BIN=1200_1ker_200k_8km

Table 15: Single-kernel design, $N_k = 200$, $N_M = 8$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 1200
/common/definitions.h	#define NKGRID 300 #define NKM_GRID 8
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 0 #define _WITHIN_ECONOMY 1 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=1ker_300k_8km HOST_BIN=1200_1ker_300k_8km

Table 16: Single-kernel design, $N_k = 300$, $N_M = 8$, 1200 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 120
/common/definitions.h	#define NKGRID 100 #define NKM_GRID 4
/common/dev_options.h	#define _BASELINE 1 #define _PIPELINE 0 #define _WITHIN_ECONOMY 0 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=baseline_1ker_100k_4km HOST_BIN=120_1ker_100k_4km

Table 17: Single-kernel Baseline design, $N_k = 100$, $N_M = 4$, 120 Economies.

File	Modify
/common/app.cpp	#define N_MODEL 120
/common/definitions.h	#define NKGRID 100 #define NKM_GRID 4
/common/dev_options.h	#define _BASELINE 0 #define _PIPELINE 1 #define _WITHIN_ECONOMY 0 #define _ACROSS_ECONOMY 0
/fpga/design.cfg	### single kernel [connectivity] nk=runOnfpga:1:runOnfpga_1
In the terminal	tmux make clean unset XCL_EMULATION_MODE source $AWS_FPGA_REPO_DIR/vitis_setup.sh export PLATFORM_REPO_PATHS=$(dirname $AWS_PLATFORM) export XCL_EMULATION_MODE=hw make afi FPGA_BIN=pipeline_1ker_100k_4km HOST_BIN=120_1ker_100k_4km

Table 18: Single-kernel only-pipelining design, $N_k = 100$, $N_M = 4$, 120 Economies.