Programming FPGAs for Economics: An Introduction to Electrical Engineering Economics

Bhagath Cheela [1] **Andre DeHon**[1]
Jesús Fernández-Villaverde[2] **Alessandro Peri** [3]

[1]University of Pennsylvania, Electrical and Systems Engineering,

[2]University of Pennsylvania, Economics

[3]University of Colorado Boulder, Economics

ASSA Annual Meeting, January 2023

What we do

- We show how to use FPGAs and their HLS compilers to solve Krusell Smith (1998)
- **Amazon Web Services**:
    - **Speedup**: Acceleration of one single FPGA is comparable to 78 CPU cores
    - **Costs Savings**: <18% of multi-core CPU acceleration
    - **Energy Savings**: <5% of multi-core CPU acceleration
- **Speed Gains**: pipeline, data-level parallelism, and data precision

What we do

- We show how to use FPGAs and their HLS compilers to solve Krusell Smith (1998)
- **Amazon Web Services**:
  - **Speedup**: Acceleration of one single FPGA is comparable to 78 CPU cores
  - **Costs Savings**: $<$18% of multi-core CPU acceleration
  - **Energy Savings**: $<$5% of multi-core CPU acceleration
- **Speed Gains**: pipeline, data-level parallelism, and data precision

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

CPU/GPUs

- Application Specific Integrated Circuit

FPGAs

- Application Specific Integrated Circuit

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

CPU/GPUs

- Application Specific Integrated Circuit
- 3GHz/1GHz

FPGAs

- Application Specific Integrated Circuit
- 250MHz

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

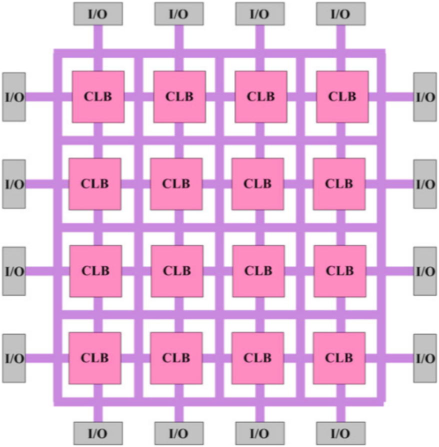
CPU/GPUs

- Application Specific Integrated Circuit
- 3GHz/1GHz
- Designed to efficiently execute serial (graphical) operations

FPGAs

- Application Specific Integrated Circuit
- 250MHz
- **Fully programmable**

Field-Programmable Gate Arrays

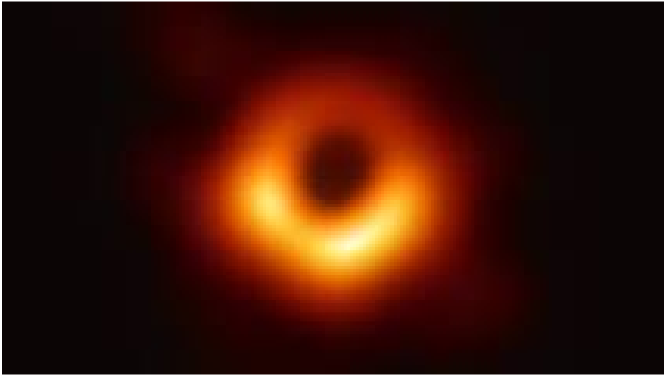


How do we get the most out of our scarce computational resources? We specialize

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

- **Research:** DNA matching (Hoang, 1993), molecular dynamics (Azizi et al., 2004), Basic Local Alignment Search Tool (BLAST) (Herbordt et al., 2006), astrophysics particles simulator (Berczik et al., 2009), cancer treatment (Young-Schultz et al., 2020)



- **Finance:** JP Morgan Estimation Risk Parameters of Derivative Portfolio

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

- **Research:** DNA matching (Hoang, 1993), molecular dynamics (Azizi et al., 2004), Basic Local Alignment Search Tool (BLAST) (Herbordt et al., 2006), astrophysics particles simulator (Berczik et al., 2009), cancer treatment (Young-Schultz et al., 2020)
- **Finance:** JP Morgan Estimation Risk Parameters of Derivative Portfolio
- **Economics**: RBC Model (Peri, 2020)...RTL approach

How do we get higher performance than a processor while retaining programmability?

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

- **Research:** DNA matching (Hoang, 1993), molecular dynamics (Azizi et al., 2004), Basic Local Alignment Search Tool (BLAST) (Herbordt et al., 2006), astrophysics particles simulator (Berczik et al., 2009), cancer treatment (Young-Schultz et al., 2020)
- **Finance:** JP Morgan Estimation Risk Parameters of Derivative Portfolio
- **Economics**: RBC Model (Peri, 2020). . . RTL approach

How do we get higher performance than a processor
while retaining programmability?

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

- **Research:** DNA matching (Hoang, 1993), molecular dynamics (Azizi et al., 2004), Basic Local Alignment Search Tool (BLAST) (Herbordt et al., 2006), astrophysics particles simulator (Berczik et al., 2009), cancer treatment (Young-Schultz et al., 2020)
- **Finance:** JP Morgan Estimation Risk Parameters of Derivative Portfolio
- **Economics**: RBC Model (Peri, 2020)... RTL approach

How do we get higher performance than a processor while retaining programmability?

Field-Programmable Gate Arrays

We show how to use **FPGAs** and their HLS compilers to solve Krusell Smith (1998)

- **Research:** DNA matching (Hoang, 1993), molecular dynamics (Azizi et al., 2004), Basic Local Alignment Search Tool (BLAST) (Herbordt et al., 2006), astrophysics particles simulator (Berczik et al., 2009), cancer treatment (Young-Schultz et al., 2020)
- **Finance:** JP Morgan Estimation Risk Parameters of Derivative Portfolio
- **Economics**: RBC Model (Peri, 2020)... RTL approach

How do we get higher performance than a processor while retaining programmability?

HLS Compilers

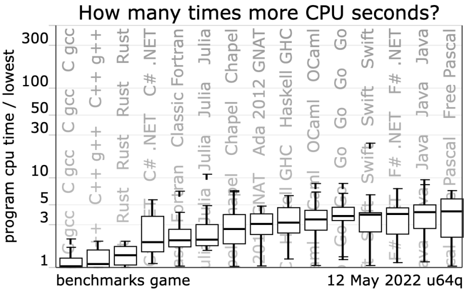
We show how to use FPGAs and their **HLS compilers** to solve Krusell Smith (1998)

- We illustrate the use of Xilinx high-level synthesis FPGA C-to-gates
  - We code our solution in C/C++ (Aruoba and Fernndez-Villaverde, 2015)
  - `#PRAGMAs` instruct the compiler on how to design the FPGA hardware
  - HLS compilers are bound to get easier and easier to use

HLS Compilers

We show how to use FPGAs and their **HLS compilers** to solve Krusell Smith (1998)

- We illustrate the use of Xilinx high-level synthesis FPGA C-to-gates
    - We code our solution in C/C++ (Aruoba and Fernndez-Villaverde, 2015)



    - `#PRAGMAs` instruct the compiler on how to design the FPGA hardware

HLS Compilers

We show how to use FPGAs and their **HLS compilers** to solve Krusell Smith (1998)

- We illustrate the use of Xilinx high-level synthesis FPGA C-to-gates
  - We code our solution in C/C++ (Aruoba and Ferndez-Villaverde, 2015)
  - `#PRAGMAs` instruct the compiler on how to design the FPGA hardware
  - HLS compilers are bound to get easier and easier to use

HLS Compilers

We show how to use FPGAs and their **HLS compilers** to solve Krusell Smith (1998)

- We illustrate the use of Xilinx high-level synthesis FPGA C-to-gates
  - We code our solution in C/C++ (Aruoba and Fernndez-Villaverde, 2015)
  - `#PRAGMAs` instruct the compiler on how to design the FPGA hardware
  - HLS compilers are bound to get easier and easier to use

Application

We show how to use FPGAs and their HLS compilers to solve **Krusell Smith (1998)**

- Heterogenous agent models with incomplete markets and aggregate uncertainty. Den Haan and Rendahl (2010)
- Solution algorithm. Maliar et al. (2010)
- Acceleration techniques can be generalized.
- **Software.** Rust (1997), Algan et al. (2008), Reiter (2009), Den Haan and Rendahl (2010), Maliar et al. (2010), Reiter (2010), Young (2010), Algan et al. (2014), Sager (2014) Pröhl (2015), Nuño and Thomas (2016), Achdou et al. (2021), Bhandari et al. (2017), Brumm and Scheidegger (2017), Judd et al. (2017), Bayer and Luetticke (2018), Childers (2018), Mertens and Judd (2018), Winberry (2018), Fernández-Villaverde et al. (2019), Auclert et al. (2020), Bilal (2021), Kahou et al. (2021)
- **Hardware.** Aldrich et al. (2011), Duarte et al. (2019), Peri (2020)

Application

We show how to use FPGAs and their HLS compilers to solve **Krusell Smith (1998)**

- Heterogenous agent models with incomplete markets and aggregate uncertainty. Den Haan and Rendahl (2010)
- Solution algorithm. Maliar et al. (2010)
- Acceleration techniques can be generalized.
- **Software.** Rust (1997), Algan et al. (2008), Reiter (2009), Den Haan and Rendahl (2010), Maliar et al. (2010), Reiter (2010), Young (2010), Algan et al. (2014), Sager (2014) Pröhl (2015), Nuño and Thomas (2016), Achdou et al. (2021), Bhandari et al. (2017), Brumm and Scheidegger (2017), Judd et al. (2017), Bayer and Luetticke (2018), Childers (2018), Mertens and Judd (2018), Winberry (2018), Fernández-Villaverde et al. (2019), Auclert et al. (2020), Bilal (2021), Kahou et al. (2021)
- **Hardware.** Aldrich et al. (2011), Duarte et al. (2019), Peri (2020)

Application

We show how to use FPGAs and their HLS compilers to solve **Krusell Smith (1998)**

- Heterogenous agent models with incomplete markets and aggregate uncertainty. Den Haan and Rendahl (2010)

- Solution algorithm. Maliar et al. (2010)

- Acceleration techniques can be generalized.

- **Software.** Rust (1997), Algan et al. (2008), Reiter (2009), Den Haan and Rendahl (2010), Maliar et al. (2010), Reiter (2010), Young (2010), Algan et al. (2014), Sager (2014) Pröhl (2015), Nuño and Thomas (2016), Achdou et al. (2021), Bhandari et al. (2017), Brumm and Scheidegger (2017), Judd et al. (2017), Bayer and Luetticke (2018), Childers (2018), Mertens and Judd (2018), Winberry (2018), Fernández-Villaverde et al. (2019), Auclert et al. (2020), Bilal (2021), Kahou et al. (2021)

- **Hardware.** Aldrich et al. (2011), Duarte et al. (2019), Peri (2020)

Application

We show how to use FPGAs and their HLS compilers to solve **Krusell Smith (1998)**

- Heterogenous agent models with incomplete markets and aggregate uncertainty. Den Haan and Rendahl (2010)
- Solution algorithm. Maliar et al. (2010)
- Acceleration techniques can be generalized.
- **Software.** Rust (1997), Algan et al. (2008), Reiter (2009), Den Haan and Rendahl (2010), Maliar et al. (2010), Reiter (2010), Young (2010), Algan et al. (2014), Sager (2014) Pröhl (2015), Nuño and Thomas (2016), Achdou et al. (2021), Bhandari et al. (2017), Brumm and Scheidegger (2017), Judd et al. (2017), Bayer and Luetticke (2018), Childers (2018), Mertens and Judd (2018), Winberry (2018), Fernández-Villaverde et al. (2019), Auclert et al. (2020), Bilal (2021), Kahou et al. (2021)
- **Hardware.** Aldrich et al. (2011), Duarte et al. (2019), Peri (2020)

Application

We show how to use FPGAs and their HLS compilers to solve **Krusell Smith (1998)**

- Heterogenous agent models with incomplete markets and aggregate uncertainty. Den Haan and Rendahl (2010)
- Solution algorithm. Maliar et al. (2010)
- Acceleration techniques can be generalized.
- **Software.** Rust (1997), Algan et al. (2008), Reiter (2009), Den Haan and Rendahl (2010), Maliar et al. (2010), Reiter (2010), Young (2010), Algan et al. (2014), Sager (2014) Pröhl (2015), Nuño and Thomas (2016), Achdou et al. (2021), Bhandari et al. (2017), Brumm and Scheidegger (2017), Judd et al. (2017), Bayer and Luetticke (2018), Childers (2018), Mertens and Judd (2018), Winberry (2018), Fernández-Villaverde et al. (2019), Auclert et al. (2020), Bilal (2021), Kahou et al. (2021)
- **Hardware.** Aldrich et al. (2011), Duarte et al. (2019), Peri (2020)

The Model

Equilibrium Calibration

- **Individual Agents Problem (IAP)**

$$\max_{\{c_t, k_{t+1}\}_{t=0}^{\infty}} \sum_{t=0}^{\infty} \beta^t \, \mathbb{E}_0 \left[\frac{c_t^{1-\gamma} - 1}{1-\gamma} \right]$$

$$\text{s.t. } c_t + k_{t+1} = \left[\mu(1-\epsilon_t) + (1-\tau_t)\bar{l}\epsilon_t \right] w_t + (1 - \delta + r_t) k_t$$

$$k_{t+1} \geq 0$$

- **Representative Firm Problem**

$$Y_t = A_t(\bar{l}L_t)_t^{1-\alpha} K_t^{\alpha}$$

$$r_t = \alpha A_t \left(\frac{\bar{l}L_t}{K_t} \right)^{1-\alpha} \qquad w_t = (1-\alpha) A_t \left(\frac{K_t}{\bar{l}L_t} \right)^{\alpha}$$

- **Government**: $\tau_t \bar{l} L_t = \mu(1 - L_t)$

- **Aggregate Law of Motion**: $\mathbf{\Gamma}_{t+1} = \mathcal{H}(\mathbf{\Gamma}_t, A_t, A_{t+1})$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method
2. **Simulation.** At each period $t = 1, \dots, 1,100$:
   - *Accumulation Step.* $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$
   - *Interpolation Step.* $k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t)$
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a)\ln m_t + \nu_t,\ t \in \{101, \dots, 1100\}$

- Update $b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1-\eta_b) b_l^i(a), \quad l \in \{1,2\}, \quad a \in \{a_b, a_g\}$
- Repeat 1-3 until convergence: $\sqrt{\sum_{l\in\{1,2\}, a\in\{a_b,a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method
2. **Simulation.** At each period $t = 1, \ldots, 1,100$:
   - *Accumulation Step.* $$m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$$
   - *Interpolation Step.* $k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t)$
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a)\ln m_t + \nu_t,\ t \in \{101, \ldots, 1100\}$

- Update $$b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1-\eta_b) b_l^i(a), \quad l \in \{1,2\}, \quad a \in \{a_b, a_g\}$$
- Repeat 1-3 until convergence: $$\sqrt{\sum_{l\in\{1,2\},a\in\{a_b,a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method — Interpolation
2. **Simulation.** At each period $t = 1, \ldots, 1, 100$:
   - *Accumulation Step.* $$m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$$
   - *Interpolation Step.* $k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t)$ — Interpolation
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a)\ln m_t + \nu_t,\ t \in \{101, \ldots, 1100\}$

- Update $$b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1-\eta_b) b_l^i(a), \quad l \in \{1,2\}, \quad a \in \{a_b, a_g\}$$
- Repeat 1-3 until convergence: $$\sqrt{\sum_{l\in\{1,2\}, a\in\{a_b,a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method — Interpolation
2. **Simulation.** At each period $t = 1, \dots, 1, 100$:
   - *Accumulation Step.* $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$ — Accumulation
   - *Interpolation Step.* $k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t)$ — Interpolation
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a)\ln m_t + \nu_t,\ t \in \{101, \dots, 1100\}$

- Update $$b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1-\eta_b) b_l^i(a), \quad l \in \{1,2\}, \quad a \in \{a_b, a_g\}$$
- Repeat 1-3 until convergence: $$\sqrt{\sum_{l\in\{1,2\}, a\in\{a_b,a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$$

Acceleration Schemes and Hardware Architecture

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

Boulder

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

CPU Multi-core Acceleration

- **CPU-C Kernel**: 3x as fast as Matlab
  - **Algorithm**: Fast interpolation range search algorithm
  - **Compilers**: `G++ 9.4.0` and `mpiCC 4.1.1` (`OpenMPI`)
  - **Optimization flags**: `-O3`
- **Amazon M5N**: 1 (m5n.large), 8 (m5n.4xlarge), 48 (m5n.24xlarge) core(s)
- **Open-MPI workflow**:
  - collect available cores
  - spread (data-independent) economies across the cores
  - 1200 economies (Robustness: `loadbalance`)

FPGA Acceleration

- **Amazon F1**: 1 (f1.2xlarge), 2 (f1.4xlarge), 8 (f1.16xlarge) FPGA(s)
- **Workflow:**
  - host **initializes** parameters, grids, guesses
  - host **launches** jobs across available FPGAs (`OpenCL`)
  - **Kernel**: FPGA(s) solve(s) the algorithm (*Custom Logic Hardware Design*)
  - host **reads back** and saves the results (`OpenCL`)

FPGA Acceleration

- **Amazon F1**: 1 (f1.2xlarge), 2 (f1.4xlarge), 8 (f1.16xlarge) FPGA(s)
- **Workflow:**
  - host **initializes** parameters, grids, guesses
  - host **launches** jobs across available FPGAs (`OpenCL`)
  - **Kernel**: FPGA(s) solve(s) the algorithm (*Custom Logic Hardware Design*)
  - host **reads back** and saves the results (`OpenCL`)

FPGA Acceleration

- **Amazon F1**: 1 (f1.2xlarge), 2 (f1.4xlarge), 8 (f1.16xlarge) FPGA(s)
- **Workflow:**
  - host **initializes** parameters, grids, guesses
  - host **launches** jobs across available FPGAs (`OpenCL`)
  - **Kernel**: FPGA(s) solve(s) the algorithm (*Custom Logic Hardware Design*)
  - host **reads back** and saves the results (`OpenCL`)

FPGA Acceleration

- **Amazon F1**: 1 (f1.2xlarge), 2 (f1.4xlarge), 8 (f1.16xlarge) FPGA(s)
- **Workflow:**
  - host **initializes** parameters, grids, guesses
  - host **launches** jobs across available FPGAs (`OpenCL`)
  - **Kernel**: FPGA(s) solve(s) the algorithm (*Custom Logic Hardware Design*)
  - host **reads back** and saves the results (`OpenCL`)

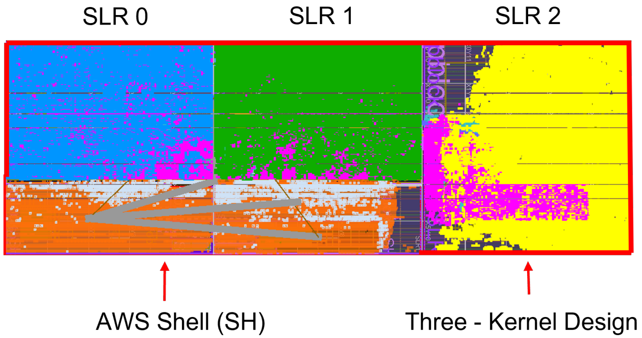
FPGA Acceleration

- **Amazon F1**: 1 (f1.2xlarge), 2 (f1.4xlarge), 8 (f1.16xlarge) FPGA(s)
- **Workflow:**
  - host **initializes** parameters, grids, guesses
  - host **launches** jobs across available FPGAs (`OpenCL`)
  - **Kernel**: FPGA(s) solve(s) the algorithm (*Custom Logic Hardware Design*)
  - host **reads back** and saves the results (`OpenCL`)

Hardware Design

Custom Logic Hardware Design

- Compute three economies (**kernels**) in parallel (one per SLR)



Boulder

Kernel Design

- **Common Challenges and Remedies**
  - Global memory access latency — Local Memories
    - Global memory large but slow (tens of clock cycles)
    - On-chip local memories small, but numerous and fast (single clock)
  - Memory access constraints: two ports for reading or writing — Make Copies
    `#pragma HLS ARRAY_PARTITION`
- **Application-Specific Challenges and Remedies**
  - Linear Interpolation — Pipelined jump search algorithm
  - **Accumulation Step.** $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$ — Fixed-precision
    - Floating-point addition: non-associative (Example) multiple clock cycles

Kernel Design

- **Common Challenges and Remedies**
  - Global memory access latency — Local Memories
    - Global memory large but slow (tens of clock cycles)
    - On-chip local memories small, but numerous and fast (single clock)
  - Memory access constraints: two ports for reading or writing — Make Copies
    `#pragma HLS ARRAY_PARTITION`
- **Application-Specific Challenges and Remedies**
  - Linear Interpolation — Pipelined jump search algorithm
  - **Accumulation Step.** $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$ — Fixed-precision
    - Floating-point addition: non-associative (Example) multiple clock cycles

Kernel Design

- **Common Challenges and Remedies**
  - Global memory access latency — Local Memories
    - Global memory large but slow (tens of clock cycles)
    - On-chip local memories small, but numerous and fast (single clock)
  - Memory access constraints: two ports for reading or writing — Make Copies
    `#pragma HLS ARRAY_PARTITION`
- **Application-Specific Challenges and Remedies**
  - Linear Interpolation — Pipelined jump search algorithm
  - **Accumulation Step.** $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$ — Fixed-precision
    - Floating-point addition: non-associative (Example) multiple clock cycles

Kernel Design

- **Common Challenges and Remedies**
  - Global memory access latency — Local Memories
    - Global memory large but slow (tens of clock cycles)
    - On-chip local memories small, but numerous and fast (single clock)
  - Memory access constraints: two ports for reading or writing — Make Copies
    `#pragma HLS ARRAY_PARTITION`
- **Application-Specific Challenges and Remedies**
  - Linear Interpolation — Pipelined jump search algorithm
  - **Accumulation Step.** $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$ — Fixed-precision
    - Floating-point addition: non-associative (Example) multiple clock cycles

Efficiency Gains

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

Cores	Speedup			Relative Costs (%)			Energy (%)		
	FPGAs			*FPGAs*			*FPGAs*		
	1	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

Speedup

- 1 FPGA performance of **78.49 cores**.
- 8 FPGAs performance of 604.38 cores.

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

	Speedup			Relative Costs (%)			Energy (%)		
	FPGAs			*FPGAs*			*FPGAs*		
Cores	*1*	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

Speedup

- 1 FPGA performance of 78.49 cores.
- 8 FPGAs performance of **604.38 cores**.

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

	Speedup			Relative Costs (%)			Energy (%)		
	FPGAs			*FPGAs*			*FPGAs*		
Cores	*1*	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

Costs

- Costs $=$ Total Execution Time $\times$ AWS on-demand prices
- FPGA acceleration solves at less than **18.35%** of the CPU cost
- One million economies: \$1043 $\rightarrow$ \$184

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

	Speedup			Relative Costs (%)			Energy (%)		
	FPGAs			*FPGAs*			*FPGAs*		
Cores	*1*	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

Costs

- Costs = Total Execution Time $\times$ AWS on-demand prices
- FPGA acceleration solves at less than 18.35% of the CPU cost
- One million economies: \$1043 $\rightarrow$ \$184

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

Cores	Speedup			Relative Costs (%)			Energy (%)		
	FPGAs			*FPGAs*			*FPGAs*		
	1	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

Energy

- Energy $=$ Total Execution time $\times$ Power

CPU core (8Watts), FPGA (33Watts)

- FPGA Energy is 5.46% of CPU Energy

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

	Speedup			Relative Costs (%)			Energy (%)		
	FPGAs			*FPGAs*			*FPGAs*		
Cores	*1*	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

- Organizations with in-house computational clusters *Departments, Central Banks*
    - Relax power limits constraints
    -

Performance

▸ Time Performance

Table: Efficiency Gains of FPGA Acceleration

	Speedup			**Relative Costs (%)**			**Energy (%)**		
Cores	*FPGAs*			*FPGAs*			*FPGAs*		
	1	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
1	78.49	156.38	604.38	17.67	17.73	18.35	5.26	5.28	5.46
8	11.00	21.91	84.68	15.76	15.82	16.37	4.69	4.71	4.87
48	1.67	3.32	12.83	17.34	17.40	18.01	5.16	5.18	5.36

- Organizations with in-house computational clusters *Departments, Central Banks*
  - Relax power limits constraints
  - Clusters are expensive to maintain (HPC specialist, $\$85,000$)

Robustness

- **Robustness**
  - Single Kernel Design: 37x faster ▸ Link
  - Performance increasing in Grids Size: up to 100+ faster than single core ▸ Link
  - Inspecting the Mechanism: pipeline, data parallelism ▸ Link
- **Carbon Footprint** of Summit and Blanca Research Computing ▸ Link
  - 150,000,000 CPU hours: 838.78 Metric Tons of CO_2 **168 cars per year**
  - 1,911,071 FPGA hours: 27.12 Metric Tons of CO_2 **5 cars per year**

Conclusions

- FPGA and HLS compiler to solve heterogeneous agent models
- With minor modifications of C-code we document:
    - speedup of the magnitude of medium-to-high scale clusters
    - costs savings ($<$18.35 %)
    - energy savings ($<$5.46 %) (reduction of carboon footprint)
- Tutorial (85 pages)
- **Next Steps:** Climate Change Models, HANK Models, chips

Conclusions

- FPGA and HLS compiler to solve heterogeneous agent models
- With minor modifications of C-code we document:
  - speedup of the magnitude of medium-to-high scale clusters
  - costs savings ($<$18.35 %)
  - energy savings ($<$5.46 %) (reduction of carboon footprint)
- Tutorial (85 pages)
- **Next Steps:** Climate Change Models, HANK Models, chips

Conclusions

- FPGA and HLS compiler to solve heterogeneous agent models
- With minor modifications of C-code we document:
  - speedup of the magnitude of medium-to-high scale clusters
  - costs savings ($<$18.35 %)
  - energy savings ($<$5.46 %) (reduction of carboon footprint)
- Tutorial (85 pages)
- **Next Steps:** Climate Change Models, HANK Models, chips

Conclusions

- FPGA and HLS compiler to solve heterogeneous agent models
- With minor modifications of C-code we document:
  - speedup of the magnitude of medium-to-high scale clusters
  - costs savings (<18.35 %)
  - energy savings (<5.46 %) (reduction of carboon footprint)
- Tutorial (85 pages)
- **Next Steps:** Climate Change Models, HANK Models, chips

Conclusions

- FPGA and HLS compiler to solve heterogeneous agent models
- With minor modifications of C-code we document:
    - speedup of the magnitude of medium-to-high scale clusters
    - costs savings (<18.35 %)
    - energy savings (<5.46 %) (reduction of carboon footprint)
- Tutorial (85 pages)
- **Next Steps:** Climate Change Models, HANK Models, chips

ASICs


Steve Jobs iPhone 2007 Presentation (HD)
"People who are really serious about software should make their own hardware."
– Alan Kay

Extra Material

The Algorithm

1 **Individual Households' Problem (IHP)**

- Policy Function Iteration
- Endogenous Grid Method

2 **Aggregate Law of Motion**

3 **Simulation** Stochastic Simulation

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$u'(c)\mathrm{d}k' = \mathbb{E}\left[(1-\delta+r')u'(c') \,|\, \epsilon, A\right] \mathrm{d}k'$$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$u'(c)\mathrm{d}k' \qquad \geq \qquad \mathbb{E}\left[(1-\delta+r')u'(c') \,|\, \epsilon, A\right] \mathrm{d}k'$$

$$\text{Borrowing Constraint :} \qquad k' \geq 0 \qquad \lambda k' = 0$$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$u'(c)\mathrm{d}k' \quad = \quad \lambda + \mathbb{E}\left[(1-\delta+r')u'(c') \,|\, \epsilon, A\right] \mathrm{d}k'$$

Borrowing Constraint : $\qquad k' \geq 0 \qquad \lambda k' = 0$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$c \quad = \quad u'^{,-1}\left(\lambda + \mathbb{E}\left[(1-\delta+r')u'(c') \,|\, \epsilon, A\right]\right)$$

Borrowing Constraint : $\qquad k' \geq 0 \qquad \lambda k' = 0$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$\underbrace{\text{Wealth}(k) - k'}_{\text{Consumption}} \quad = \quad u'^{,-1}\left(\lambda + \mathbb{E}\left[(1-\delta+r')u'(c') \,|\, \epsilon, A\right]\right)$$

$$\text{Borrowing Constraint :} \qquad k' \geq 0 \qquad \lambda k' = 0$$

$$\text{Wealth}(k) = \text{Wealth}(k, \epsilon, m, A) = \left(\mu(1-\epsilon) + (1-\tau)\bar{l}\epsilon\right) w + (1-\delta+r)k$$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$\underbrace{\text{Wealth}(k) - k'}_{\text{Consumption}} = u'^{,-1}\left(\lambda + \mathbb{E}\left[(1-\delta+r')u'\underbrace{\left(\text{Wealth}(k') - k''\right)}_{\text{Consumption}'} \mid \epsilon, A\right]\right)$$

$$\text{Borrowing Constraint :} \qquad k' \geq 0 \qquad \lambda k' = 0$$

$$\text{Wealth}(k) = \text{Wealth}(k, \epsilon, m, A) = (\mu(1-\epsilon) + (1-\tau)\bar{l}\epsilon)\, w + (1-\delta+r)k$$
$$\text{Wealth}(k') = \text{Wealth}(k', \epsilon', m', A') = (\mu(1-\epsilon') + (1-\tau')\bar{l}\epsilon')\, w' + (1-\delta+r')k'$$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$\underbrace{\text{Wealth}(k) - k'}_{\text{Consumption}} = u'^{,-1}\left(\lambda + \mathbb{E}\left[(1-\delta+r')u'\underbrace{\left(\text{Wealth}(k') - k''\right)}_{\text{Consumption}'} \mid \epsilon, A\right]\right)$$

$$\text{Borrowing Constraint :} \qquad k' \geq 0 \qquad \lambda k' = 0$$

$$\text{Wealth}(k) = \text{Wealth}(k, \epsilon, m, A) = \left(\mu(1-\epsilon) + (1-\tau)\bar{l}\epsilon\right) w + (1-\delta+r)k$$

$$\text{Wealth}(k') = \text{Wealth}(k', \epsilon', m', A') = \left(\mu(1-\epsilon') + (1-\tau')\bar{l}\epsilon'\right) w' + (1-\delta+r')k'$$

$$k'' \equiv k'(k') \equiv k'(k'(k, \epsilon, m, A), \epsilon', m', A')$$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$k' = \text{Wealth}(k) - u'^{,-1}\left(\lambda + \mathbb{E}\left[(1-\delta+r')u'\left(\underbrace{\text{Wealth}(k') - k''}_{\text{Consumption}'}\right)\right]\right)$$

$$\text{Borrowing Constraint :} \qquad k' \geq 0 \qquad \lambda k' = 0$$

$$\begin{aligned}
\text{Wealth}(k) = \text{Wealth}(k, \epsilon, m, A) &= \left(\mu(1-\epsilon) + (1-\tau)\bar{l}\epsilon\right) w + (1-\delta+r)k \\
\text{Wealth}(k') = \text{Wealth}(k', \epsilon', m', A') &= \left(\mu(1-\epsilon') + (1-\tau')\bar{l}\epsilon'\right) w' + (1-\delta+r')k' \\
k'' \equiv k'(k') &\equiv k'(k'(k, \epsilon, m, A), \epsilon', m', A')
\end{aligned}$$

Individual Households' Problem (IHP)

- For all states, $(k, \epsilon, m, A) \in \mathbf{K} \times \{0,1\}_\epsilon \times \mathbf{M} \times \mathbf{A}$:

$$\hat{k}' = \underbrace{\left[\mu(1-\epsilon) + (1-\tau)\bar{l}\epsilon\right] \; w + (1-\delta+r)k}_{\text{Wealth}(k,\epsilon,m,A)}$$

$$- \left\{ \lambda + \beta \mathbb{E}\left[\frac{1-\delta+r'}{\left(\underbrace{(\mu(1-\epsilon') + (1-\tau')\bar{l}\epsilon')\, w' + (1-\delta+r')k'}_{\text{Wealth}(k',\epsilon',m',A')} - k'' \right)^{\gamma}} \right] \right\}^{-1/\gamma}$$

$$k'' \equiv k'(k') \equiv k'(k'(k,\epsilon,m,A), \epsilon', m', A')$$

- Guess $k'(k, \epsilon, m, A)$.
- Set the lagrange multiplier $\lambda(k, \epsilon, m, A) = 0$
- Update $$k'_{i+1} = \eta \widehat{k}'_{i+1} + (1-\eta) k'_i$$

  until convergence $$\max_{(k,\epsilon,m,A)\in \mathbf{K}\times\{0,1\}_\epsilon\times\mathbf{M}\times\mathbf{A}} |k'_{i+1} - k'_i| < \varepsilon_k$$

Aggregate Law of Motion

- Households' distribution over capital holdings and employment status

$$\mathbf{\Gamma}' = \mathcal{H}(\mathbf{\Gamma}, A, A').$$

- Restriction 1: Set of moments, $m \in \mathbf{M}$

$$m' = H(m, A, A')$$

- Restriction 2: m is the first moment (per capita stock of capital)

$$m_t = \frac{1}{J} \sum_{j=1}^{J} k_{j,t}$$

- Restriction 3:

$$\mathbb{E}[\ln m' | a, m] = b_1(a) + b_2(a) \ln m \qquad a \in \{a_b, a_g\},$$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method
2. **Simulation.** At each period $t = 1, \ldots, 1, 100$:
   - *Accumulation Step.* $$m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$$
   - *Interpolation Step.* $$k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t),$$
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a)\ln m_t + \nu_t,\ t \in \{101, \ldots, 1100\}$
   - Update $$b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1-\eta_b) b_l^i(a), \quad l \in \{1,2\}, \quad a \in \{a_b, a_g\}$$
   - Repeat 1-3 until convergence: $$\sqrt{\sum_{l\in\{1,2\}, a\in\{a_b,a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method
2. **Simulation.** At each period $t = 1, \ldots, 1, 100$:
   - *Accumulation Step.* $$m_t = \frac{1}{J} \sum_{j=1}^{J} k_{j,t}$$
   - *Interpolation Step.* $k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t)$,
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a) \ln m_t + \nu_t,\ t \in \{101, \ldots, 1100\}$

- Update $b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1 - \eta_b) b_l^i(a), \quad l \in \{1, 2\}, \quad a \in \{a_b, a_g\}$
- Repeat 1-3 until convergence: $\sqrt{\sum_{l \in \{1,2\}, a \in \{a_b, a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method
2. **Simulation.** At each period $t = 1, \ldots, 1,100$:
   - *Accumulation Step.* $m_t = \frac{1}{J}\sum_{j=1}^{J} k_{j,t}$
   - *Interpolation Step.* $k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t)$,
3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a)\ln m_t + \nu_t,\ t \in \{101, \ldots, 1100\}$

- Update $$b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1-\eta_b) b_l^i(a), \quad l \in \{1,2\}, \quad a \in \{a_b, a_g\}$$
- Repeat 1-3 until convergence: $$\sqrt{\sum_{l\in\{1,2\}, a\in\{a_b,a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$$

Algorithm

1. **Individual Agents Problem (IAP)**
   - Policy Function Iteration
   - Endogenous Grid Method

Interpolation

2. **Simulation.** At each period $t = 1, \dots, 1, 100$:
   - *Accumulation Step.* $$m_t = \frac{1}{J} \sum_{j=1}^{J} k_{j,t}$$
   - *Interpolation Step.* $$k_{j,t+1}(k_{j,t}, \epsilon_{j,t}, m_t, A_t),$$

3. **Aggregate Law of Motion:** $\ln m_{t+1} = b_1(a) + b_2(a) \ln m_t + \nu_t,\ t \in \{101, \dots, 1100\}$

- Update $$b_l^{i+1}(a) = \eta_b \hat{b}_l^i(a) + (1 - \eta_b) b_l^i(a), \quad l \in \{1, 2\}, \quad a \in \{a_b, a_g\}$$
- Repeat 1-3 until convergence: $$\sqrt{\sum_{l \in \{1,2\}, a \in \{a_b, a_g\}} (b_l^{i+1}(a) - b_l^i(a))^2} < \varepsilon_b = 1e(-8)$$

Calibration

Back

Table: Calibrated Parameters

α	0.36	Output capital share
β	0.99	Quarterly subjective discount factor
γ	1	Arrow-Pratt relative risk aversion coefficient
δ	0.025	Quarterly depreciation rate
μ	0.15	Unemployment benefits in terms of wages
$\bar{l}$	0.9	Time endowment
Δ_A	0.01	Aggregate productivity shock size

Table: Technical Specifications

AWS Instance	**Cores**	**FPGAs**	**Pricing** ($/hour)	**Memory** (GiB)
m5n.large	1	-	0.119	8
m5n.4xlarge	8	-	0.952	64
m5n.24xlarge	48	-	5.712	384
f1.2xlarge	1	1	1.650	122
f1.4xlarge	4	2	3.300	244
f1.16xlarge	32	8	13.200	976

Table: Resource Utilization by Grids Size

Individual Capital, N_k	*100*	*200*	*300*
BRAM(%)	18.33	20.97	24.72
DSP(%)	66.92	66.92	66.92
Registers(%)	30.65	30.51	30.76
LUT(%)	67.53	68.88	70.35
URAM(%)	18.33	18.33	18.33

	CPU cores		
N.	1	8	48
Time (s)	37854.52	5303.73	803.63
Cost ($)	1.25	1.40	1.28
Energy (J)	302836.16	339438.72	308593.92
AWS Instance	m5n.large	m5n.4xlarge	m5n.24xlarge
	FPGA devices		
N.	1	2	8
Time (s)	482.30	242.06	62.63
Cost ($)	0.22	0.22	0.23
Energy (J)	15915.90	15975.96	16534.32
AWS Instance	f1.2xlarge	f1.4xlarge	f1.16xlarge

Back

Table: Time Performance by Individual Capital Grid Size, N_k

: CPU Execution Time

$N_k = 100$		$N_k = 200$		$N_k = 300$	
CPU-Cores		*CPU-Cores*		*CPU-Cores*	
8	*48*	*8*	*48*	*8*	*48*
5303.73	803.63	9502.63	1500.15	15432.15	2347.69

: FPGA Execution Time

$N_k = 100$			$N_k = 200$			$N_k = 300$		
FPGAs			*FPGAs*			*FPGAs*		
1	*2*	*8*	*1*	*2*	*8*	*1*	*2*	*8*
482.30	242.06	62.63	671.28	336.54	86.64	1057.53	529.75	134.78

Back

Equilibrium

Back

Given an exogenous transition law for $\{A, \epsilon\}$, a recursive competitive equilibrium is the set of prices $\{w, r\}$, policy function $k'(\cdot)$, tax rate τ, and law of motion $\mathcal{H}(\cdot)$ for the cross-sectional distribution $\mathbf{\Gamma}$ such that:

- given the individual household state $\{k, \epsilon; \mathbf{\Gamma}, A\}$, prices $\{w, r\}$ and the laws of motion of $\{A, \epsilon\}$ and Γ, the policy function $k'(\cdot)$ solves the Bellman equation representation of the household's sequential problem;
- given $\{\mathbf{\Gamma}, A\}$, input factor prices $\{w, r\}$ receive their marginal products;
- given A, the labor income tax rate τ balances the government budget;
- the markets for labor and capital clear;
- given $\{w, r, \mathbf{\Gamma}, k'\}$ and the transition laws for $\{A, \varepsilon\}$, the law of motion $\mathcal{H}(\cdot)$ is satisfied.

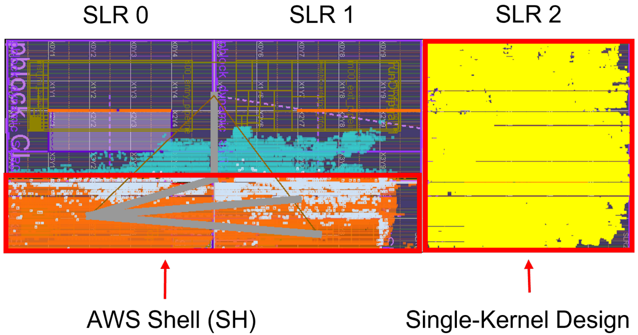
Golbderg (1991) Back

- Let
  - $x = 1e30$
  - $y = -1e30$
  - $z = 1$
- $(x + y) + z = 1$, $x + (y + z) = 0$
- Floating-point addition is non-associative

Within Economy Resources

Back

Figure: Single-kernel Design: Resource Utilization

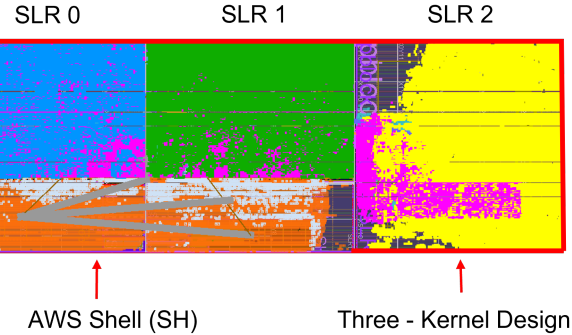


Note: Resources utilized by: *(i)* the single-kernel CL design (yellow area); *(ii)* by the AWS Shell (orange area); and *(iii)* available CL resources (other colors). The image is captured using Xilinx Vivado.

Within Economy Resources

Back

Figure: Three-kernel Design: Resource Utilization



Note: Resources utilized by: *(i)* the three-kernel CL design (yellow, green, blue areas each corresponding to one kernel); *(ii)* by the AWS Shell (orange area); and *(iii)* available CL resources (other colors, of which the pink area serves as a wrapper). The image is created using Xilinx Vivado.

Robustness

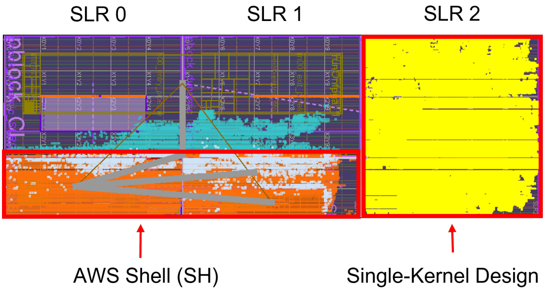
Back

Single Kernel Design

Back

Table: Speedup Comparison One-Kernel Single FPGA vs. Single CPU Core

FPGA-Time(sec)	*CPU*-Time(sec)	*Speedup*(x)	*Costs*(%)	*Energy*(%)
0.84	31.54	37.66	36.81	7.30



Performance by Grids Size

Time Performance Back

Table: Speedup Comparison across Grid Sizes

Individual Capital, N_k	*100*	*200*	*300*
1 FPGA vs. 8 Cores	11.00	14.16	14.59
2 FPGA vs. 8 Cores	21.91	28.24	29.13
8 FPGA vs. 8 Cores	84.68	109.68	114.50

Note: Speedups recorded by comparing the solution of 1,200 economies using AWS instances connected to 1, 2, and 8 FPGAs and using Open-MPI parallelization on AWS instances with 8 and 48 cores (rows) for different individual household capital N_k.

Inspecting the Mechanism

Back

Table: Speedup Gains: Acceleration Channels Accounting

	Baseline	*Pipelining*	*Data Parallelism*	
			Within Economy	*Across Econ.*
Single-core Execution / FPGA Solution	0.40			
CL Utilization (%)				
BRAM	5.48			
DSP	6.13			
Registers	3.94			
LUT	6.11			
URAM	5.50			

- Solution in 80 seconds (vs 30 seconds in CPU)
- Automatic optimization (3GHz/250MHz=14x)

Table: Speedup Gains: Acceleration Channels Accounting

	Baseline	*Pipelining*	*Data Parallelism*	
			Within Economy	*Across Econ.*
Single-core Execution				
FPGA Solution	0.40	0.57		
CL Utilization (%)				
BRAM	5.48	8.45		
DSP	6.13	12.87		
Registers	3.94	5.24		
LUT	6.11	9.14		
URAM	5.50	5.50		

Pipelining

- Interpolation
- Data precision

Table: Speedup Gains: Acceleration Channels Accounting

	Baseline	*Pipelining*	*Data Parallelism*	
			Within Economy	*Across Econ.*
Single-core Execution				
FPGA Solution	0.40	0.57	37.66	
CL Utilization (%)				
BRAM	5.48	8.45	22.26	
DSP	6.13	12.87	31.13	
Registers	3.94	5.24	12.03	
LUT	6.11	9.14	25.17	
URAM	5.50	5.50	5.50	

- Resources single-kernel design: Figure

Table: Speedup Gains: Acceleration Channels Accounting

	Baseline	*Pipelining*	*Data Parallelism*	
			Within Economy	*Across Econ.*
Single-core Execution / FPGA Solution	0.40	0.57	37.66	78.49
CL Utilization (%)				
BRAM	5.48	8.45	22.26	18.33
DSP	6.13	12.87	31.13	66.92
Registers	3.94	5.24	12.03	30.65
LUT	6.11	9.14	25.17	67.53
URAM	5.50	5.50	5.50	18.33

- Resources three-kernel design: ▸ Figure

Carbon Footprint of Research Computing

Back

- **CPU core power**: 0.013 kWh
- Xcel Energy: 37% (Natural Gas), 26% (Coal), 37% (Renewables)
- US EPA: 0.91 (Natural Gas), 2.21 (Coal), 0.1 (Renewables)
- lbs CO_2 per Xcel Colorado kWh: 0.9483lbs
- lbs CO_2 per CURC HPC core: 0.0123lbs CO_2/core hour
- Summit and Blanca Super computers: 150 millions core hours per year
    - Lbs CO_2 per year: 1,849,185 lbs
    - Metric Tons of CO_2 per year: 838.78

168 cars per year

- **FPGA power**: 0.033 kWh
- lbs CO_2 per FPGA core: 0.031 lbs CO_2/FPGA hour
- Summit and Blanca Super computers: 1,911,071 FPGA hours per year (78.49x)
    - Lbs CO_2 per year: 59,804 lbs
    - Metric Tons of CO_2 per year: 27.12

5 cars per year

Carbon Footprint of Research Computing

Back

- **CPU core power**: 0.013 kWh
- Xcel Energy: 37% (Natural Gas), 26% (Coal), 37% (Renewables)
- US EPA: 0.91 (Natural Gas), 2.21 (Coal), 0.1 (Renewables)
- lbs CO_2 per Xcel Colorado kWh: 0.9483lbs
- lbs CO_2 per CURC HPC core: 0.0123lbs CO_2/core hour
- Summit and Blanca Super computers: 150 millions core hours per year
  - Lbs CO_2 per year: 1,849,185 lbs
  - Metric Tons of CO_2 per year: 838.78

168 cars per year

- **FPGA power**: 0.033 kWh
- lbs CO_2 per FPGA core: 0.031 lbs CO_2/FPGA hour
- Summit and Blanca Super computers: 1,911,071 FPGA hours per year (78.49x)
  - Lbs CO_2 per year: 59,804 lbs
  - Metric Tons of CO_2 per year: 27.12

5 cars per year

Bibliography

Achdou, Y., J. Han, J.-M. Lasry, P.-L. Lions, and B. Moll (2021). Income and wealth distribution in macroeconomics: A continuous-time approach. *Review of Economic Studies 89*(1), 45–86.

Aldrich, E. M., J. Fernández-Villaverde, A. Ronald Gallant, and J. F. Rubio-Ramírez (2011). Tapping the supercomputer under your desk: Solving dynamic equilibrium models with graphics processors. *Journal of Economic Dynamics and Control 35*(3), 386–393.

Algan, Y., O. Allais, and W. J. Den Haan (2008). Solving heterogeneous-agent models with parameterized cross-sectional distributions. *Journal of Economic Dynamics and Control 32*(3), 875–908.

Algan, Y., O. Allais, W. J. Den Haan, and P. Rendahl (2014). Solving and simulating models with heterogeneous agents and aggregate uncertainty. In *Handbook of Computational Economics*, Volume 3, pp. 277–324. Elsevier.

Auclert, A., M. Rognlie, and L. Straub (2020). Micro jumps, macro humps: Monetary policy and business cycles in an estimated HANK model. Working Paper 26647, National Bureau of Economic Research.