A

A Hardware Approach to Value Function Iteration

Alessandro Peri
University of Colorado Boulder

June 13, 2019

A Hardware Approach to Value Function Iteration[I]

June 13, 2019

Latest Version: Link

Alessandro Peri

University of Colorado Boulder

Abstract

We propose a novel approach for the computation of dynamic stochastic equilibrium models. We design an FPGA specialized in the computation of a bellman equation via value function

computational power. In order to cope with this technological challenge, major corporations - like Google, Intel and Microsoft - have recently stopped relying on commodity general purpose hardware (i.e. CPUs) and have instead started developing their own chips. In 2017, Google presented its custom accelerator for machine learning applications: the Tensor Processing Unit (TPU). Using the words of a distinguished hardware engineer at Google: \This is roughly equivalent to fast-forwarding technology about seven years into the future (three generations of Moore's Law)".[1]

The main contribution of this paper is to bring this *hardware* approach to the macroeconomists' table. To this end, we propose a class of chips, whose hardware is explicitly designed to be fully customizable by the user: the Field-programmable gate array (FPGA).

Following Aldrich et al. [2011], we illustrate the potential of this technology in a standard RBC model. We design an FPGA specialized in the solution of a bellman equation via value function iteration (FPGA approach). In doing so, we present a *novel* parallelization scheme (the assembly line algorithm) and compare the speed gains vis-a-vis the GPU data-parallelization scheme proposed in Aldrich et al. [2011] (GPU approach). Our work documents 10-fold speed gains in the solution of the bellman equation via value function iteration, on a state space with 65536 and 4 points in the capital and productivity shock grids, respectively.

The speed gains are a byproduct of hardware specialization. FPGAs are integrated circuits with (billions of) tiny transistors, organized in con gurable logic blocks (CLBs)[2]. Di erently from CPUs and GPUs, the connections among FPGA's transistors are not pre-designed by the manufacturer. On the contrary, hardware developers can fully customize the routing network between the CLBs[3] to accelerate their target application. In this paper, we specialize the FPGA's hardware to solve a bellman equation via value function iteration. This approach

[1] \Google supercharges machine learning tasks with TPU custom chip.", Jouppi [2016].

[2] See Appendix A for an overview of the FPGA, CPU and GPU platforms.

[3] Hardware developers can describe an FPGA image using description languages like VHDL (VHSIC Hardware description language) or Verilog.

grants access to two layers of parallelism, unaccessible to software developers: *i)* instruction-level and *ii)* pipeline parallelism at the logical (CLB) resources level. Intuitively speaking, the former determines the operations to be performed in parallel at every clock cycle, while the latter organizes their synchronous execution along an assembly line. Inspired by its analogy to Ford's assembly line, we call this parallelization scheme the assembly line parallelism[4].

The FPGA approach has proven extremely successful in a variety of sectors and elds[5], from genomics, medicine[6], physics[7] to banking and nance[8]. The chip is indeed particularly suited for practical research purposes. First, conditional on knowing the hardware design principles[9], it is easily programmable[10]

on the cloud, facilitating the sharing of FPGA's algorithms. For instance, the interested reader can deploy our FPGA solution, by launching in Amazon AWS the Amazon Machine Image (AMI): FPGA - Value Function Iteration Accelerator[11].

Like the fall in the cost of DNA sequence has fostered breakthrough discoveries in genomics and medicine, the fall in the entry costs to the FPGA technology provides an important venue for the research and development of high-speed algorithms in macroeconomics. Fostered by the recent development of heterogenous agents-model featuring nominal rigidities (Bayer et al. [2019]) and a growing attention to the distributional e ect of government policies, the FPGA approach could nd a promising area of application in the complex (and sometimes unfeasible) estimation of heterogenous agents model. This paper provides a rst step in this direction, discussing the acceleration of one of the most expensive computational bottlenecks involved in this process: the solution of a bellman equation.

By and large, these considerations suggest the presence of signi cant gains from hardware specialization, so far unexplored by the macroeconomic literature.

2. Contribution

The optimization problem of agents with rational preferences rests at the heart of virtually every macroeconomic model. In this context, the bellman equation (Bellman, 1957) is a powerful tool that simpli es complex in nite-horizon maximization problems into two-period problems of the form

$$V(x;z) = T(V) = \max_{x' \in \Gamma(x;z)} F(x;z;x') + \int_{z' \in Z} V(x';z')Q(dz';z) \tag{1}$$

where agents observe the state $(x;z) \in X \times Z$ and choose a control $x' \in \Gamma(x;z)$ to maximize current $F(x;z;x')$ and expected discounted payo s $\int_Z V(x';z')Q(dz';z)$.

[11] AWS Link: https://aws.amazon.com/marketplace/pp/B07PLWCNCV

Yet, solving a Bellman Equation is computationally challenging. Under suitable assumptions - described in the Banach Contraction Mapping Theorem (see, Stokey et al. [1989]) - it is possible to approach arbitrarily close the limit function V, starting from any guess $V_0(\ )$ and iterating over the operator $T^n V_0(\ )$,

$$(T^n V_0; V$$

via value function iteration, owing to their rich endowments of processing units (ranging from 100 to 5000 cores). In their seminal paper, Aldrich et al. [2011] document speed gains akin to parallelizations on small scale super-computers.

This paper builds a bridge between these two approaches: we propose a chip (the FPGA) that allows the implementation of (software) algorithms at the hardware level.

The rest of the paper is organized as follows. Section 3 lays out the model. Section 4 discusses the computational algorithm. Section 5 introduces the assembly line parallelism. Section 6 presents the results and examines the origins of the speed gains: the assembly line parallelism. Section 7 discusses applications and extensions. Section 8 concludes.

3. The Model

In the spirit of Aldrich et al. [2011], we illustrate the FPGA approach in the context of a real business cycle model. The representative household chooses consumption c and capital k^{0} to solve the bellman equation

$$V(k;z) = \max_{c;k^{0}} \frac{c^{1}}{1} + \int_{z^{0}} V(k^{0};z^{0})Q(z^{0};z)\mathrm{d}z^{0} \qquad (2)$$

$$\text{s.t. } c + k^{0} = zk + (1 \quad)k$$

where the log-productivity z follows an $AR(1)$ 62 Tfs th Tf oe

Table 1: Parameters' Estimates

Parameter	Value	Description
	0.984	Subjective discount factor
	2.000	Arrow-Pratt relative risk aversion coe cient
	0.350	RTS parameter
	0.010	Depreciation rate
	0.950	Persistence of the AR(1) log-productivity process
	0.005	Volatility of the innovation of the AR(1) log-productivity process

Note: Parameters are set to match features of the U.S. quarterly data.

shock = 0:005. Table 1 summarizes the parameters.

4. The Computational Algorithm

We design an FPGA to solve the bellman equation in (2)

Second, the FPGA solves the value function iteration step i

$$V^i(k;z) = \max_{k' \in K} F(k;z;k') + \sum_{z'} V^{i-1}(k';z')Q(z';z)dz'$$

up to convergence of the value function $||V^i - V^{i-1}|| < \varepsilon$. In particular,

1. Given V_0, the FPGA uses the binary search algorithm discussed in Section 4.0.1

history of previous binary-stages' maximizers h $(n \quad 1)$.

5. The Assembly Line Parallelism

Each value function iteration step i

$$V^{i}(k;z) = \max_{k' \in \{z\}} F(k;z;k') + \sum_{z'} V^{i \ 1}(k';z')Q(z'$$

Peak-Finding

and *ii)* do not provide the FPGA's hardware exibility necessary for syncronizing the parallelism at the *logical resources (CLB)* level[18].

Table 2 reports the results. The FPGA results are obtained using a 16 nm Xilinx UltraScale Plus mounted on a node with Intel(R) Xeon(R) CPU E5-2686 v4 2.30GHz, 8 CPUs/node. The GPU results are obtained using an NVIDIA Tesla K80 with 4992 cores mounted on a node with Intel Xeon CPU E5-2680 v3 2.50GHz, 2 CPUs/node.

Our simulations record signi cant speed gains: the FPGA is 9.64 times faster than the GPU (13.69 vs 1.42) in solving the bellman equation (C.2) via VFI using the binary search algorithm. For completeness, we also report the time required to initialize and read back the results from the chips. The GPU outperforms the FPGA in this context. If we factor in these components the FPGA is approximately 6 times faster. Importantly, the network interconnection between the host and guest machines (FPGA and GPU) are similar, so future developments in this dimension are likely to drastically reduce this di erence[19]. While we acknowledge the presence of important margins of improvement in this dimension, we also take the occasion to stress that the nal goal of this paper is to introduce the FPGA technology and to illustrate its potential for accelerating the solution's time of a bellman equation.

The speed gains arise from the possibility of synchronizing the parallelism at the *logical resources (CLB)* level. As a result, although GPU's cores are up to three times faster than our FPGA's clock, the gains from hardware specialization more than o set the lower clock speed. The next session explores the determinants of the speed gains, by investigating the assembly line parallelism work ow.

[18] Although GPUs allow to program the pipeline parallelism at the *binary stage* level, they do not allow so at the CLBs level. The GPU's compiler takes care of this level of detail.

[19] The di erence that we observe arises from a low performing algorithm in the communication with the FPGA. In particular, the C code copies and reads information from and to the FPGA element by element rather than in blocks.

6.1. The Assembly Line Parallelism Work ow

This section exploits the analogy between the pipeline algorithm and Ford's assembly line to illustrate the assembly line parallelism work ow.

Figure 2: The assembly line parallelism work ow

Note: This gure illustrates the assembly line work ow associated4.045 1 101/ac0(w)hure

de ned as the set of registers connected to the master signal.

Figure 2

of risk models to account for systemic risk. In 2011, JP Morgan directly tackled the problem. After a rst attempt to accelerate their estimation via GPUs (with a 15-fold speed-up), they hired a specialized company to take advantage of FPGA chips. The new platform reduced the execution time by a factor of 130, from 10 hours to 4 minutes. More far-reaching, the gains of relaxing the computational feasibility constraint went beyond the faster execution of the estimation. Traders were able to change the parameterization of the model to assess di erent scenarios and better inform their investment decisions.

This case study shows important micro-prudential policy implications of FPGA acceleration. Similarly, the growing attention to the distributional e ect of government policies - fostered by the recent development of heterogenous agents-model featuring nominal rigidities (Bayer et al. [2019]) - suggests that the FPGA approach could nd a promising area of application in the complex (and sometimes unfeasible) estimation of heterogenous agents models[22].

8. Conclusions and Extensions

In the last decades, the use of highly-specialized chips (like, FPGAs and ASICs) has become pervasive across several sectors and elds. From bitcoin mining to machine learning accelerators, there has been a growing acknowledgment of the gains arising from hardware specialization. The main contribution of this paper is to bring this \hardware" approach to the macroeconomists' table.

To this end, we design a Field-programmable gate array (FPGA) specialized in the solution of bellman equations via value function iteration. Our hardware approach proposes a

[22] Although, these models provide a powerful tool to quantitatively assess the e ect of government policies on rms and households' distributions, the richness of predictions is hindered by their computationally expensive and time consuming estimations. The global search over the parameters' space may require solving the bellman equation several times (even in the order of millions), restricting the richness of heterogeneity that can be studied. In addition, as the number of parameters to be estimated rise, the estimation slips into the curse of dimensionality problem and may become unfeasible.

novel parallelization scheme and documents signi cant speed gains vis-a-vis GPU-based data-parallelization techniques. The speed gains are a byproduct of the hardware specialization. The hardware exibility grants access to two levels of parallelism, which are unavailable to software developers: *i)* the instruction-level and *ii)* pipeline parallelism at the logical resources level. We label this parallelization scheme as the *assembly line parallelism*.

This paper highlights the presence of important gains from FPGA acceleration in the context of heterogenous agents model, which we plan to explore in our future research. Importantly, the plethora of computational issues in economics which could bene t from FPGA-acceleration goes beyond the solution of bellman equations. Since FPGAs are best suited to handle large scale computational problems whose solutions are computationally intensive and exploit a recursive algorithm, machine learning, montecarlo simulations, and maximum likelihood estimation seem promising areas of application.

To conclude, we would like to discuss a venue that could signi cantly reduce the cost of access to the FPGA technology: the use of FPGA compiler specialized in translating C/Matlab codes into customized digital circuits[23]. Although the use of FPGA compilers is not yet the standard in the FPGA designers world, recent years have shown signs of early adoption. Drawing from the successful experience of C compilers in replacing the direct assembler programming of CPU, major developments in FPGA compilers could drammatically reduce the cost of access to the FPGA acceleration in the years to come.

D. Michalik, A. Montana, W. Montgomerie, M. Mora-Klein, D. Muders, A. Nadolski, S. Navarro, C. H. Nguyen, H. Nishioka, T. Norton, G. Nystrom, H. Ogawa, P. Oshiro, T. Oyama, S. Padin, H. Parsons, S. N. Paine, J. Penalver, N. M. Phillips, M. Poirier, N. Pradel, R. A. Primiani, P. A. Ra n, A. S. Rahlin, G. Reiland, C. Risacher, I. Ruiz, A. F. Saez-Mada n, R. Sassella, P. Schellart, P. Shaw, K. M. Silva, H. Shiokawa, D. R. Smith, W. Snow, K. Souccar, D. Sousa, T. K. Sridharan, R. Srinivasan, W. Stahm, A. A. Stark, K. Story, S. T. Timmer, L. Vertatschitsch, C. Walther, T.-S. Wei, N. Whitehorn, A. R. Whitney, D. P. Woody, J. G. A. Wouterloot, M. Wright, P. Yamaguchi, C.-Y. Yu, M. Zeballos, and L. Ziurys (2019). First M87 Event Horizon Telescope Results. II. Array and Instrumentation. *The Astrophysical Journal 875*(1), L2.

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

Aruoba, S. B., J. Fernandez-Villaverde, and J. F. Rubio-Ram rez (2006). Comparing solution methods for dynamic equilibrium economies. *Journal of Economic Dynamics and Control 30*(12), 2477{2508.

Auclert, A., R. Matthew, and L. Straub (2019). Micro Jumps, Macro Humps: Monetary Policy and Business Cycles in an Estimated HANK Model. *Unpublished working paper*.

Bayer, C., R. Luetticke, L. Pham-Dao, and V. Tjaden (2019). Precautionary Savings, Illiquid Assets, and the Aggregate Consequences of Shocks to Household Income Risk. *Econometrica 87*(1), 255{290.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press.

Bhandari, A., D. Evans, M. Golosov, and T. J. Sargent (2017). Fiscal policy and debt management with incomplete markets. *Quarterly Journal of Economics 132*(2), 617{663.

Carroll, C. D. (2006). The method of endogenous gridpoints for solving dynamic stochastic optimization problems. *Economics Letters 91*(3), 312{320.

Dan elsson, J. (2002). The emperor has no clothes: Limits to risk modelling. *Journal of Banking and Finance 26*(7), 1273{1296.

Dan elsson, J. (2008). Blame the models. *Journal of Financial Stability 4*(4), 321{328.

De Schryver, C. (2015). *FPGA Based Accelerators for Financial Applications*.

Farooq, U., Z. Marrakchi, and H. Mehrez (2012). Tree-based heterogeneous FPGA architectures: Application speci c exploration and optimization. In *Springer, New York, NY*, Volume 9781461435, pp. 1{186.

Feldman, M. (2011). JP Morgan Buys Into FPGA Supercomputing. *HPCwire, https://www.hpcwire.com/2011/07/13/jp_morgan_buys_into_fpga_supercomputing/*.

Fella, G. (2014). A generalized endogenous grid method for non-concave problems. *Review of Economic Dynamics 17*(2), 329{344.

Fernandez-Villaverde, J. and D. Z. Valencia (2018). A Practical Guide to Parallelization in Economics. *NBER Working Paper No 24561*, 1{65.

Frittelli, M., M. Maggis, and I. Peri (2014). Risk measures on P(R) and value at risk with probability/loss function. *Mathematical Finance 24*(3), 442{463.

Gordon, G. and S. Qiu (2015). A Divide and Conquer Algorithm for Exploiting Policy Function Monotonicity. *SSRN Electronic Journal*.

Jouppi, N. (2016). Google supercharges machine learning tasks with TPU custom chip. *O - cial Google Blog, https://cloud.google.com/blog/products/gcp/google-supercharges-machine-learning-tasks-with-custom-chip*.

Margulies, M., M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y. J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, X. V. Gomes, B. C. Godwin, W. He, S. Helgesen, C. H. Ho, G. P. Irzyk, S. C. Jando, M. L. Alenquer, T. P. Jarvie, K. B. Jirage, J. B. Kim, J. R. Knight, J. R. Lanza, J. H. Leamon, S. M. Lefkowitz, M. Lei, J. Li, K. L. Lohman, H. Lu, V. B. Makhijani, K. E. McDade, M. P. McKenna, E. W. Myers, E. Nickerson, J. R. Nobile, R. Plant, B. P. Puc, M. T. Ronan, G. T. Roth, G. J. Sarkis, J. F. Simons, J. W. Simpson, M. Srinivasan, K. R. Tartaro, A. Tomasz, K. A. Vogt, G. A. Volkmer, S. H. Wang, Y. Wang, M. P. Weiner, P. Yu, R. F. Begley, and J. M. Rothberg (2005). Genome sequencing in microfabricated high-density picolitre reactors. *Nature 437*(7057), 376{380.

Saxena, A. (2014). Oscars host's sel e tweet crashes Twitter. *Website: https://www.gadgetsnow.com/social/Oscars-hosts-sel*.

Stokey, N. L., R. E. Lucas, and E. C. Prescott (1989). *Recursive Methods in Economic Dynamics*. Harvard University Press.

Tauchen, G. (1986). Finite state markov-chain approximations to univariate and vector autoregressions. *Economics Letters 20*(2), 177{181.

T-396(v)T-31 72/s7an11aE.v

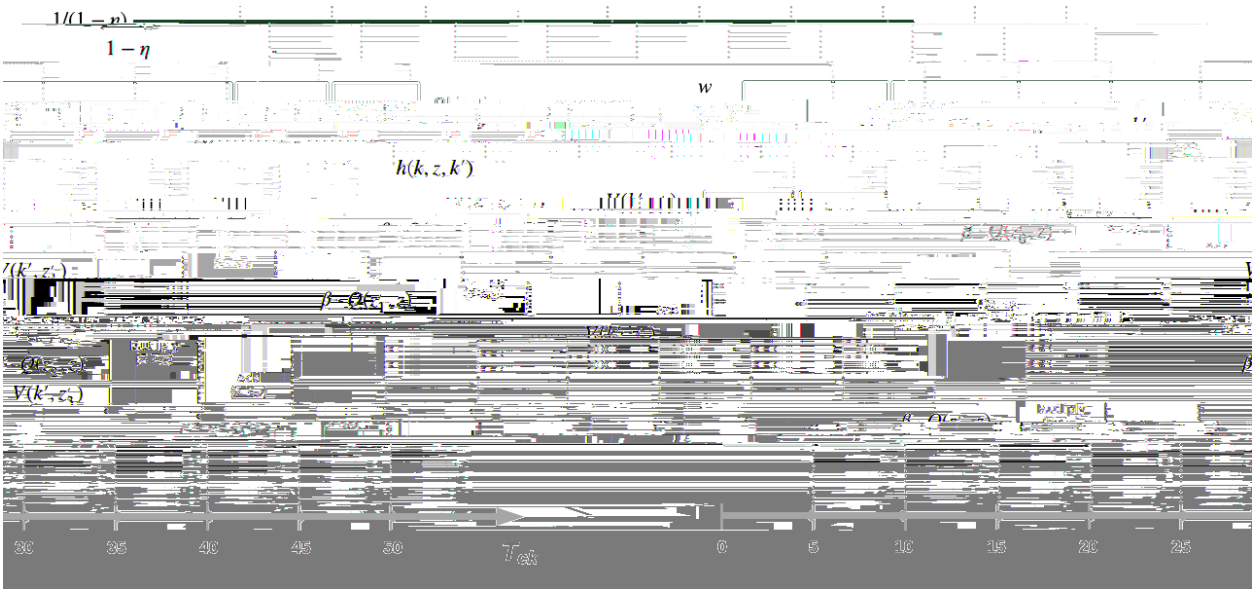
Appendix A. An overview of the FPGA, CPU and GPU chips

Like CPUs and GPUs, eld programmable gate arrays (FPGAs) are integrated circuits (IC) with (billions of) tiny transistors. Di erently from CPUs and GPUs, the transistors in the FPGA are organized in con gurable logic blocks (CLBs) and their connections are not pre-designed by the manufacturer, but can be programmed

Appendix B.4. Step 4: Using the FPGA image in Amazon AWS.

For instructions on how to use our FPGA - Value Function Iteration Accelerator the interest reader can refer

Figure C.4: The Instruction Level Parallelism



Note: The Figure illustrates the instruction-level parellelism involved in the computation of the objective function, given the state $(k; z)$ and control k^0. The horizontal dimension denotes the evolution of time. The algorithm parellelizes the computation of the return function (top half) and continuation term (bottom half), taking as given: 1) the parameters $(,)$; 2) the wealth $w(k; z)$; 3) the value functions $V(k^0; z_i^0)$ and transition matrix $Q(z_i^0; z$

Table C.3: Map of Operations in the Objective Functions

	Adds/ Subs	Multiplicators	Logarithm	Exponential
$F(k; z; k^0)$	1	2	1	1
$\max_{z^0} V(k^0, z^0) Q(z^0, z)$	3	4	-	-
$h(k; z; k^0)$	5	6	1	1

Note

2. Returns the maximizer $j^{*} = \arg\max_{j \in \{1;2;3\}} \{h(k; z; k'(i(n; j)))\}$[26], and the associated maximizer $i(n; j^{*}) \in \{0; 1; 2; \ldots; N_k - 1\}$.

3. Selects the nth-stage indexes $\{i(n; j)\}$ in a set $G(n) \subseteq \{0; 1; \ldots; N_k - 1\}$ of feasible indexes $g(n; l) \in \{0; 1; 2; \ldots; N_k - 1\}$

$$G(n) = \left\{ g(n; l) : \{0; 1; 2; \ldots; 4 \cdot 2^{n-1} - 1\} \to \{0; 1; \ldots; N_k - 1\} \right\} \qquad g(n; l) = \left\lfloor \frac{N_k}{2^{n+1} \cdot \text{Step}(n)} \right\rfloor l \qquad \text{(C.3)}$$

according to the following selection rule

$$\{i(n; j)\}_{j \in \{1;2;3\}} = \left\{ i(n; j) : \{1; 2; 3\} \to \{0; 1; \ldots; N_k - 1\} \right\} \qquad i(n; j) = \left\lfloor \frac{N_k}{2^{n+1} \cdot \text{Step}(n)} \right\rfloor (h(n) + j) \qquad \text{(C.4)}$$

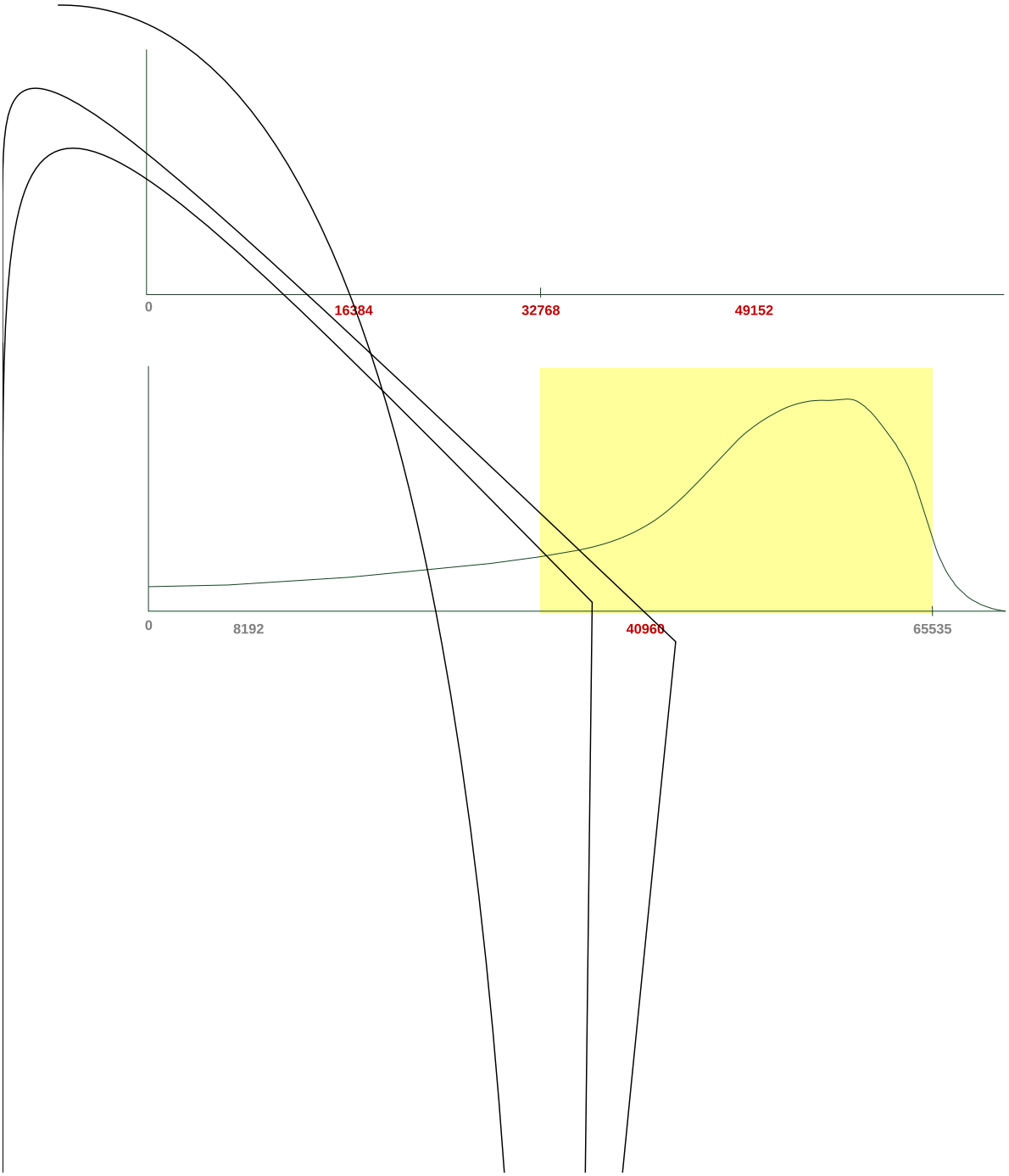
where $h(1) = 0$, and

$$h(n+1) = 2 \cdot h(n) + j^{*}(n) \qquad \text{(C.5)}$$

and where

$$j^{*}(n) = \begin{cases} 0 & \text{if } h(k; z; k'(i(n; j = 1))) \text{ is the max}^{27} \\ 1 & \text{if } h(k; z; k'(i(n; j = 1))) = h(k; z; k'(i(n; j = 2)) \text{ are the max} \\ 2 & \text{if } h(k; z; k'(i(n; j = 2))) \text{ is the max} \\ 2 & \text{if } h(k; z; k'(i(n; j = 1))) = h(k; z; k'(i(n; j = 2))) = h(k; z; k'(i(n; j \end{cases}$$

Figure C.5: Binary Search Algorithm: An Example



The grey lines in Figure C.5 illustrate graphically the evolution of the set of feasible indexes $g(n;l) \in \{0;1;\dots;65535\}$, that is the set of indexes among which the 3 nth-stage indexes $\{i(n;j)\}_{j\in\{1;2;3\}}$ (red lines) are selected at every stage n. In order to efficiently cover the search range $R(n)$ (yellow areas), the set $G(n) \in \{0;1;\dots;65535\}$

$$G(n) = \left\{ g(n;l) : l \in \{0;1;2;\dots;4 \cdot 2^{n-1} - 1\} \right\} \quad g(n;l) = \frac{N_k}{2^{n+1}} \cdot \left\lceil \frac{l}{\text{Step}(n)} \right\rceil$$

uniformly spreads the feasible indexes $g(n;l)$ (grey lines) at distance $\text{Step}(n)$ of each other. In the first stage $n = 1$: $g(1;1) = 16384$, $g(1;2) = 32768$, $g(1;3) = 49152$. As the search range $R(n)$ halves at every binary stage n (yellow areas), the set of feasible indexes gradually becomes finer: the distance between the feasible indexes halves (as captured by $\text{Step}(n)$), while the cardinality of the feasible set $|G(n)|$ doubles (as captured by $l \in \{0;1;2;\dots;4 \cdot 2^{n-1} - 1\}$).

Each binary stage n evaluates the objective function (C.1) $\{h(k;z;k'(i(n;j)))\}_{j\in\{1;2;3\}}$ at the 3 nth-stage indexes $\{i(n;j)\}_{j\in\{1;2;3\}}$ and returns the maximizer $j^* = \arg\max_{j\in\{1;2;3\}} \{h(k;z;k'(i(n;j)))\}$, and accordingly, $i(n;j^*) \in \{0;1;2;\dots;N_k - 1\}$. In our example, $i(1;j^* = 3) = 49152$, $i(2;j^* = 3) = 57344$, $i(3;j^* = 2) = 57344$,...

As emerge intuitively in Figure C.5, the selection rule of the nth-stage indexes (red lines)

$$\{i(n;j)\}_{j\in\{1;2;3\}} = \left\{ i(n;j) : j \in \{1;2;3\} \right\} \quad i(n;j) = \frac{N_k}{2^{n+1}} \cdot \left\lceil \frac{(h(n)+j)}{\text{Step}(n)} \right\rceil$$

depends on both the evolution of the set of feasible indexes $G(n)$ (grey lines)

$$G(n) = \left\{ g(n;l) : l \in \{0;1;2;\dots;4 \cdot 2^{n-1} - 1\} \right\} \quad g(n;l) = \frac{N_k}{2^{n+1}} \cdot \left\lceil \frac{l}{\text{Step}(n)} \right\rceil$$

and the history of maximizers

$$h(n+1) = 2 \cdot h(n) + j^*(n)$$

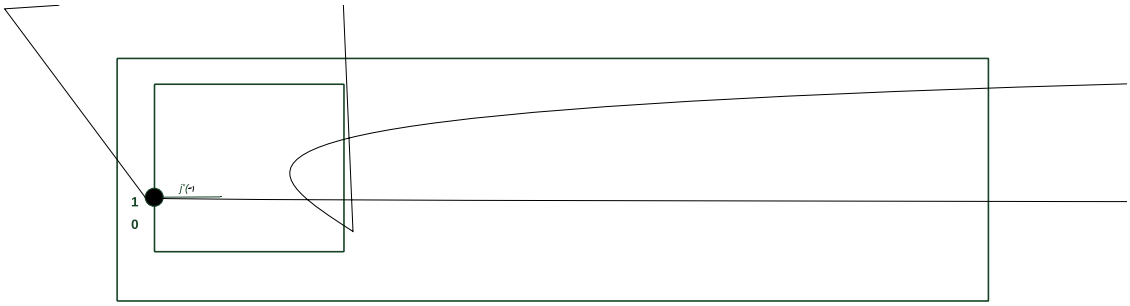
where $h(1) = 0$ and $j^*(n)$ is described by (C.6). In general, $h(n)$ provides a sufficient statistic for the

The implementation phase exploits the recursive structure of this binary search algorithm to build up the pipeline parallelism. The next session discusses the details.

Appendix C.3. The Assembly Line Parallelism

The pipeline parallelism exploits the recursive structure of the binary search algorithm to organize the binary stages around an assembly line, as illustrated in Figure C.6.

Figure C.6: The Assembly Line Parallelism



Note: The gure illustrates the organization of the 15 binary stages along the assembly line. Given the state $(z;k)$, the previous stage maximizer $j\ (n\ \ 1)$ and a su cient statistic of the history of previous-stages maximizers $h\ (n\ \ 1)$, each binary-stage updates $j\ (n)$ and $h\ (n)$. After 15 stages, the assembly line returns the value function $V(k;z)$ and maximiser $i\ (k;z)$.

For the sake of exposition, let us think at each binary stage as a production team. The 15 teams (binary stages) are assigned:

1. a position $n\ 2\ f1;2;:::;14;15g$ in the assembly line;
2. a task , represented by the orange cloud in Figure C.6. To perform their task, teams need three inputs, as denoted by the three lines entering each orange cloud from the left: the state $k(z)$ and

Figure C.7: The Assembly Line Parallelism: Details

Note: The gure illustrates the organization of the Assembly Line Algorithm. The top half of the gure illustrates the 15 binary stages. The bottom half of the gure details the instructions involved at each stage over the clock cycles.

position along an assembly line, as illustrated at the bottom of Figure (C.7). In our implementation, the first 5 workers in each binary stage select the 3nth-stage indexes and access the memory, the next 50 workers evaluate the objective functions as illustrated in Figure C.4; the last 4 workers determine the stage maximum and maximiser and return the results to the next stage.