Scalable high-dimensional dynamic stochastic economic modeling

Brumm, Johannes ; Scheidegger, Simon ; Mikushin, Dmitry ; Schenk, Olaf

Abstract: We present a highly parallelizable and flexible computational method to solve high-dimensional stochastic dynamic economic models. Solving such models often requires the use of iterative methods, like time iteration or dynamic programming. By exploiting the generic iterative structure of this broad class of economic problems, we propose a parallelization scheme that favors hybrid massively parallel computer architectures. Within a parallel nonlinear time iteration framework, we interpolate policy functions partially on GPUs using an adaptive sparse grid algorithm with piecewise linear hierarchical basis functions. GPUs accelerate this part of the computation one order of magnitude thus reducing overall computation time by 50%. The developments in this paper include the use of a fully adaptive sparse grid algorithm and the use of a mixed MPI-Intel TBB-CUDA/Thrust implementation to improve the interprocess communication strategy on massively parallel architectures. Numerical experiments on "Piz Daint" (Cray XC30) at the Swiss National Supercomputing Centre show that high-dimensional international real business cycle models can be efficiently solved in parallel. To our knowledge, this performance on a massively parallel petascale architecture for such nonlinear high-dimensional economic models has not been possible prior to present work.

DOI: https://doi.org/10.1016/j.jocs.2015.07.004

Title: Scalable High-Dimensional Dynamic Stochastic Economic Modeling

Author: Johannes Brumm Dmitry Mikushin Simon Scheidegger Olaf Schenk

PII: S1877-7503(15)30005-3
DOI: http://dx.doi.org/doi:10.1016/j.jocs.2015.07.004
Reference: JOCS 393

To appear in:

Received date: 15-11-2014
Revised date: 27-7-2015
Accepted date: 30-7-2015

Please cite this article as: Johannes Brumm, Dmitry Mikushin, Simon Scheidegger, Olaf Schenk, Scalable High-Dimensional Dynamic Stochastic Economic Modeling, *Journal of Computational Science* (2015), http://dx.doi.org/10.1016/j.jocs.2015.07.004

Scalable High-Dimensional Dynamic Stochastic Economic Modeling

Johannes Brumm[a,1], Dmitry Mikushin[b,1], Simon Scheidegger[a], Olaf Schenk[b,2]

[a] *Department of Banking and Finance, Faculty of Economics, University of Zurich, Switzerland*
[b] *Institute of Computational Science, Università della Svizzera italiana, Via Giuseppe Buffi 13, CH-6900 Lugano, Switzerland*

Abstract

We present a highly parallelizable and flexible computational method to solve high-dimensional stochastic dynamic economic models. Solving such models often requires the use of iterative methods, like time iteration or dynamic programming. By exploiting the generic iterative structure of this broad class of economic problems, we propose a parallelization scheme that favors hybrid massively parallel computer architectures. Within a parallel nonlinear time iteration framework, we interpolate policy functions partially on GPUs using an adaptive sparse grid algorithm with piecewise linear hierarchical basis functions. GPUs accelerate this part of the computation one order of magnitude thus reducing overall computation time by 50%. The developments in this paper include the use of a fully adaptive sparse grid algorithm and the use of a mixed MPI-Intel TBB-CUDA/Thrust implementation to improve the interprocess communication strategy on massively parallel architectures. Numerical experiments on "Piz Daint" (Cray XC30) at the Swiss National Supercomputing Centre show that high-dimensional international real business cycle models can be efficiently solved in parallel. To our knowledge, this performance on a massively parallel petascale architecture for such nonlinear high-dimensional economic models has not been possible prior to present work.

Email addresses: johannes.brumm@uzh.ch (Johannes Brumm), dmitry.mikushin@usi.ch (Dmitry Mikushin), simon.scheidegger@uzh.ch (Simon Scheidegger), olaf.schenk@usi.ch (Olaf Schenk)

[1] Johannes Brumm gratefully acknowledges financial support from the European Research Council grant "Solving dynamic models: Theory and Applications" (PI Felix Kübler).

[2] Corresponding author.

Keywords: international real business cycles, high-dimensional grids, adaptive sparse grids, parallel algorithms, high-performance computing, heterogeneous systems

1. Introduction

Clearly, the world economy is an extremely complex system. Even when modeling only the most relevant features of a small part of this system, one easily ends up with a large and intricate formal structure. This is often due to the heterogeneity across different consumers, workers, households, firms, sectors, or countries. A further complication stems from the fact that human beings choose their actions based on expectations about an uncertain future. This feedback from the future makes dynamic stochastic economic modeling particularly difficult (see, e.g., [1, 2]). Model-based economics has for the most part reacted to this challenge in two ways. Either by focusing on qualitative results obtained from extremely simplified models with little heterogeneity, or by only locally solving the equation systems that describe the dynamics around a so-called steady state. In contrast, solving for the global solution of a model with substantial heterogeneity is very costly: The computation time and storage requirements increase dramatically with the amount of heterogeneity, i.e. with the dimensionality of the problem. It is therefore often far beyond the scope of current methods to include as much heterogeneity as a natural modeling choice would suggest. In overlapping generations models researchers often use time-steps that exceed one year by far. For instance, Krueger and Kubler [3] analyze the welfare implications of social security reform in a model where one period corresponds to six years, thereby reducing the number of adult cohorts and thus the dimensionality of the problem by a factor of six. Similarly, international real business cycle (IRBC) models often include only a very small number of countries or regions. Bengui et al. [4], for example, analyze cross-country risk-sharing at the business cycle frequency using a two country model — one 'focus' country versus the rest of the world. Reducing the dimensionality of the problem in such ways can deliver valuable qualitative insights. However, to derive solid quantitative results or even to test the robustness of the qualitative results, one often has to look at problems of higher dimension.

Building on [5], this paper shows how we can use modern numerical methods and cutting-edge supercomputing facilities to compute global solutions of high-dimensional dynamic stochastic economic models in a way that fits their generic structure. No matter whether these are solved by iterating on a Bellman equation to update a value function (*parametric dynamic programming*; see, e.g., [6, 7]) or by iterating on systems of non-linear equations that represent equilibrium conditions to update functions that represent economic choices (*time iteration*; see, e.g., [6]), the computational challenge is similar:

(i) In each iteration step, an economic function needs to be approximated. For this purpose, the function value has to be determined at many points in the high-dimensional state space, and

(ii) each point involves solving a high-dimensional maximization problem (for dynamic programming) or a system of nonlinear equations (for time iteration).

These two important features of the considered problems create difficulties in achieving a fast time-to-solution process. We overcome these difficulties by minimizing both the number of points to be evaluated and the time needed for each evaluation. For the first purpose (i) we use adaptive sparse grids (see, e.g., [8, 9]), while the second task (ii) is accomplished using a hybrid parallelization scheme that minimizes interprocess communication by using Intel Threading Building Blocks (TBB) [10] and partially offloads the function evaluations to accelerators using CUDA/Thrust [11]. This scheme enables us to make efficient use of modern hybrid high-performance computing facilities, whose performance nowadays reaches multiple petaflops [12]. Their hybrid architecture typically features CPU compute nodes with attached GPUs[3]. We show in this paper that the generic structure of an algorithm that solves dynamic economic models by time iteration or dynamic programming using sparse grids is a natural match for such hybrid systems.

[3]The Swiss National Supercomputer Centre's "Piz Daint" Cray XC30 that is used in the numerical experiments in Sec. 5 consists of Intel Xeon E5 processors with NVIDIA Tesla K20X GPUs attached to it; its peak performance is 7.7 petaflops.

Sparse grid interpolation alleviates the *curse of dimensionality* [13] faced by interpolation on standard tensor product grids: Starting with a one-dimensional discretization scheme that employs N grid points, a naive extension to d dimensions using tensor products leads to N^d grid points. In the IRBC model we consider as an application, d depends on the number of countries included in the model. Sparse grids reduce the number of grid points needed from the order $\mathcal{O}\left(N^d\right)$ to $\mathcal{O}\left(N \cdot (\log N)^{d-1}\right)$, while the accuracy of the interpolation only slightly deteriorates in the case of sufficiently smooth functions [8]. Sparse grids go back to Smolyak [14] and have been applied to a whole range of different research fields such as physics, visualization, data mining, Hamilton-Jacobi Bellman (HJB) equations, mathematical finance, insurance, and econometrics [15, 16, 8, 17, 18, 19, 20, 21, 22]. Krueger and Kubler [3] and Judd et al. [23] solve dynamic economic models using sparse grids with global polynomials as basis functions. In contrast, we use piecewise-linear local basis functions first introduced by Zenger [24] in the context of sparse grids. The hierarchical structure of these basis functions lends itself for an adaptive refinement strategy as, e.g., in Ma and Zabaras [9], Bungartz and Dirnstorfer [25], or Pflüger [26]. This adaptive grid can better capture the local behavior of functions that have steep gradients or even nondifferentiabilities. The latter feature naturally arise from occasionally binding constraints which are present in many economic models yet have so far been tractable only in low-dimensional cases [27, 28, 29].

Parallel computing and sparse grids [30, 31, 32, 33, 34, 35, 36, 37] enters the picture when we have to solve high-dimensional nonlinear equation systems (or maximization problems) at each point of the sparse grid. Fortunately, within each iteration step, these tasks are independent from each other and can thus be solved in parallel by distributing them via MPI [38] to different processes. When searching for the solution to the equation system at a given point, the algorithm has to frequently interpolate the function computed in the previous iteration step. These interpolations take up 99% of the computation time needed to solve the equation system. As they have a high arithmetic intensity, i.e., many arithmetic operations are performed for each byte of memory transfer and access, they are perfectly suited for GPUs [31, 39, 21]. We therefore offload parts of the interpolation tasks from the compute nodes to their attached accelerators,

which results in a reduction of the overall computation time by roughly 50%. Due to the indicated high intrinsic level of parallelism, the economic modeling code can efficiently use CPU-GPU hybrid supercomputing systems. Our large scale numerical experiments performed on the Piz Daint XC30 machine from the Swiss National Supercomputing Centre (CSCS) show that the developments of this paper make it possible to solve realistically sized and thus high-dimensional, heterogeneous economic models in times that are considerably under one hour. To the best of our knowledge, this has not been possible before. We also observe very good strong scaling efficiencies on Piz Daint.

Summing up, we present a method for solving a large class of generic high-dimensional dynamic stochastic economic models of size and complexity that were not tractable in a reasonable time before. Using adaptive sparse grids, we build a hybrid-parallel iterative procedure which, by construction, can efficiently use modern high-performance computing architectures. With this work, we hope to help computational research become a strong "third pillar" of economics, alongside theory and experimentation, as it already is in many sciences, like physics or chemistry. Despite the promise of high-performance computing facilities to solve complex economic models, economists have in the past been restrained in doing so. Single GPUs were used in several applications in order to accelerate computations [40], whereas Cai and Judd [41] used the high latency "Condor" paradigm to solve dynamic programming problems in parallel. However, apart from Brumm and Scheidegger [5] and Cai et al. [42], who recently exploited highly parallel low-latency systems, no one in computational economics has so far made the effort to make efficient use of the most advanced contemporary high-performance computing (HPC) systems.

The remainder of the paper is organized as follows. In Sec. 2, we describe the structure of the dynamic economic models we solve. In Sec. 3, we briefly outline the construction of adaptive sparse grids. In Sec. 4, we embed adaptive sparse grid interpolation in a time iteration algorithm. We then discuss in Sec. 5 the performance of this algorithm and report how hybrid parallelization can speed up the computations. Section 6 concludes.

2. High-dimensional dynamic economic models

To capture complex economic phenomena, models often have to include several different economic *agents* (who choose actions that are optimal given their objectives). Depending on the research question, these agents might represent firms, sectors, countries, or certain subgroups of the population, ordered by skills, age, or other characteristics. If the model is dynamic, it is common practice to consider so-called recursive equilibria [1, 43], where the state of the economy can be summarized by a *state variable* and the dynamics of the economy can be captured by a time-invariant function of this state. In most applications, the state variable contains agents' characteristics, for instance their accumulated assets. When multiple agents and/or several of their relevant characteristics are included, the state of the economy can quickly become high-dimensional. As the state influences agents' behavior and thereby the dynamics of the economic model, it is a serious challenge for numerical methods to capture these dynamics if the state space is high-dimensional. To describe this challenge more formally, we first describe the general structure common to many (infinite-horizon) dynamic economic models. In a second step, we describe one concrete example, the IRBC model (see, e.g., [44, 45]).

2.1. Dynamic economic models as functional equations

Let $x_t \in X \subset \mathbb{R}^d$ denote the state of the economy at time $t \in \mathbb{N}$. Then the actions of all agents can be represented by a *policy function* $p : X \to Y$, where Y is the space of possible *policies*. The stochastic transition of the economy from period t to $t+1$ can then be represented by the distribution of next period's state x_{t+1}, which depends on the current state and policies:

$$x_{t+1} \sim F\left(\cdot | x_t, p(x_t)\right). \tag{1}$$

While the distribution F as a function of x_t and $p(x_t)$ is implied by the economic assumptions of the model, the policy function p needs to be determined from *equilibrium conditions*. When using time iteration (see Sec. 4), these conditions include agents' first-order optimality conditions, next to other conditions like budget constraints or

market clearing [6]. Taken together, these conditions constitute a functional equation that the policy function p has to satisfy, namely, that for all $x_t \in X$,

$$0 = \mathbb{E}\left\{E\Big(x_t, x_{t+1}, p\left(x_t\right), p\left(x_{t+1}\right)\Big) | x_t, p(x_t)\right\}, \tag{2}$$

where the expectation, represented by the operator $\mathbb{E}$, is taken with respect to the distribution $F\left(\cdot|x_t, p(x_t)\right)$ of next period's state x_{t+1}. The function $E : X^2 \times Y^2 \to \mathbb{R}^{2d}$ represents the period-to-period equilibrium conditions of the model. In most economic applications, this function is nonlinear because of concavity assumptions on utility and production functions. As a consequence, the optimal policy p solving (2) will also be nonlinear, or even nonsmooth if the model includes so-called (economic) frictions, like borrowing constraints or irreversible investments. Therefore, approximating this function only locally, which is often sufficient if the model exhibits low volatility, might provide misleading results when larger fluctuations are considered. For such applications, we need a global solution, that is, we need to approximate p over the entire state space X or at least on the ergodic distribution of the stochastic state variable (see, e.g., [23]). We solve for the policy function p using a time iteration procedure (see Sec. 4). As a consequence, we need to interpolate many successive approximations of p. To do this efficiently we employ (adaptive) sparse grids (see Sec. 3). Next, we introduce the example that we apply our algorithm to. Readers who are more interested in our solution methods can skip the subsequent example.

2.2. Example: The international real business cycle model

To demonstrate the capabilities of our method, we choose the IRBC model, which has become a workhorse for studying methods for solving high-dimensional economic models [46]. The IRBC model is relatively simple and easy to explain, whereas the dimensionality of the model can be scaled up in a straightforward and meaningful way as it just depends linearly on the number of countries considered. This feature of the model allows us to focus on the problem of handling high-dimensional state spaces. To show that we can also handle nonsmooth problems, we consider a version of the IRBC model where investment is irreversible.

Stochastic dynamic equilibrium models have been used to study business cycle fluctuations of economic aggregates like output, consumption, and investment since the seminal work of Kydland and Prescott [47]. Initially, the focus was on models of closed economies with the United States being the main application. Later, these real business cycle models were extended to include several countries and were thus called IRBC models [44]. They can be used to model co-movements and spillovers across countries as well as current account deficits and exchange rate movements. However, previous research (see, e.g., [44, 4] analyzed setups with only a very small number of countries/regions (mainly two or three) and mostly considered models without occasionally binding constraints. We solve models with many more countries that nevertheless include occasionally binding constraints, in particular irreversible investment. To focus on the high-dimensionality of the state space, we keep the model simple in other ways. Extending the model in directions that are standard in the literature, however, is not a serious challenge for our solution method. For instance, to discuss exchange rate movements one would have to consider a model with several commodities, differentiated by the country in which they are produced. This extension would not increase the dimension of the state space, just the size of the equations systems that have to be solved.

In the model we are considering, there are M countries, $j = 1, \ldots, M$, each using its accumulated capital stock, k_t^j, to produce the output good, which can either be used for investment, χ_t^j, or for consumption, c_t^j, generating utility, $u^j(c_t^j)$, with constant relative risk aversion utility $u^j(c) = c^{-\gamma}/(1-\gamma)$ and risk-aversion parameter γ. Investment is subject to adjustment costs, and it is irreversible in the following sense: The capital stock of a country can neither be consumed nor used for production in another country — an assumption that seems more realistic than perfect reversibility, which is normally assumed to keep the model tractable. However, capital depreciates at a rate $\delta > 0$ and can thus nevertheless shrink over time if there is not enough new investment. Thus, the law of motion of capital is

$$k_{t+1}^j = k_t^j \cdot (1-\delta) + \chi_t^j. \tag{3}$$

The amount produced by each country is given by

$$a_t^j \cdot A \cdot (k_t^j)^\kappa.$$

It thus depends on the size of the capital stock employed, k_t^j, on the overall productivity level, A, as well as on the country specific productivity level, a_t^j, which has the following law of motion:

$$\ln a_t^j = \rho \cdot \ln a_{t-1}^j + \sigma \left(e_t^j + e_t^{M+1} \right). \tag{4}$$

The parameters ρ and σ determine persistence and volatility in productivity. The country specific shocks, $e_t^j \sim \mathcal{N}(0,1)$, as well as the global shock, $e_t^{M+1} \sim \mathcal{N}(0,1)$, are assumed to be independent from each other and across time. However, we assume that countries can insure themselves against these shocks by trading assets with payoffs that are contingent on the realization of these shocks. This complete markets assumption[4] implies that the market allocation of capital and consumption (the so-called decentralized competitive equilibrium) can be obtained as the solution to a social planner's problem: maximize the weighted sum of all countries utility from consumption, weighted by welfare weights, τ^j, which depend on the initial capital stocks of the countries. Thus, the social planner solves the following infinite-horizon problem, where the future is discounted by the discount factor, β:

$$\max_{\{c_t^j, k_t^j\}} \mathbb{E}_0 \sum_{j=1}^{M} \tau^j \cdot \left(\sum_{t=1}^{\infty} \beta^t \cdot u^j(c_t^j) \right), \tag{5}$$

subject to the aggregate resource constraint

$$\sum_{j=1}^{M} \left(a_t^j \cdot A \cdot (k_t^j)^\kappa - k_t^j \cdot \frac{\phi}{2} \cdot (g_{t+1}^j)^2 - \chi_t^j - c_t^j \right) = 0, \tag{6}$$

and the constraint that investment in each country j, χ_t^j, is irreversible,

$$\chi_t^j \geq 0. \tag{7}$$

[4]We follow the comparison study by [45] in assuming complete markets. However, we are not directly solving the social planner problem, but iterate on the period-to-period equilibrium conditions given in (8)–(10) below. With incomplete markets, these conditions have a similar structure and our method can thus be applied as well.

In (6), the first term is the amount produced by each country, while the second term represents the convex adjustment costs, where ϕ parametrizes the intensity of capital adjustment costs, and $g^j_{t+1} \equiv k^j_{t+1}/k^j_t - 1$ is the growth rate of capital in country j.

So far, we are considering an infinite horizon problem. However, as mentioned above, it is common practice in economics to focus on recursive equilibria [2, 1], where the state of the economy is summarized by a state variable and the dynamics of the economy is capture by a time-invariant function of this state. We now briefly present the recursive structure of the above IRBC model, while we refer the reader to [5] for the derivation of the equilibrium conditions. The state variables of the IRBC model with M countries consist of

$$x_t = \left(a^1_t, \ldots, a^M_t, k^1_t, \ldots, k^M_t\right) \in X \subset \mathbb{R}^{2M},$$

where a^j_t and k^j_t are the productivity and capital stock of country j, respectively. The policy function $p : \mathbb{R}^{2M} \to \mathbb{R}^{2M+1}$ maps the current state into investment choices, χ^j_t, the multipliers for the irreversibility constraints, μ^j_t, and the multiplier of the aggregate resource constraint, λ_t:

$$p(x_t) = \left(\chi^1_t, \ldots, \chi^M_t, \mu^1_t, \ldots, \mu^M_t, \lambda_t\right).$$

The investment choices determine next period's capital stock in a deterministic way through (3). In contrast, the law of motion of productivity, (4), is stochastic. Taken together, (3) and (4) specify the distribution of x_{t+1} (corresponding to (1) in the general problem above). The period-to-period equilibrium conditions of this model (corresponding to (2)) consist of three types of equations. First are the optimality conditions for investment in capital in each country j:[5]

$$\begin{aligned} &\lambda_t \cdot \left[1 + \phi \cdot g^j_{t+1}\right] - \mu^j_t \\ &-\beta \mathbb{E}_t \left\{ \lambda_{t+1} \left[a^j_{t+1} A \kappa (k^j_{t+1})^{\kappa-1} + 1 - \delta + \frac{\phi}{2} g^j_{t+2} \left(g^j_{t+2} + 2 \right) \right] - \mu^j_{t+1}(1-\delta) \right\} = 0 \end{aligned} \quad (8)$$

[5] The expectation in Eq. 8 below is given by the following integral: $(2\pi)^{-\frac{M+1}{2}} \int \Omega(x_t, e_t) \cdot \exp(-e'_t \cdot e_t/2) \cdot de_t$, where $\Omega(x_t, e_t)$ is defined as the term whithin the expectation, and $e_t = \left(e^1_t, ..., e^{M+1}_t\right)$.

Second is the irreversibility assumption for investment in each country j, and the associated complementarity conditions,

$$\chi_t^j \geq 0, \mu_t^j \geq 0, \chi_t^j \cdot \mu_t^j = 0. \tag{9}$$

Finally is the aggregate resource constraint

$$\sum_{j=1}^{M} \left(a_t^j \cdot A \cdot (k_t^j)^\kappa - k_t^j \cdot \frac{\phi}{2} \cdot (g_{t+1}^j)^2 - \chi_t^j - c_t \right) = 0, \tag{10}$$

where we can use the fact that $c_t = (\lambda_t / \tau_j)^{-\gamma^j}$ at an optimal choice.

For all the parameters of the economic model, we make standard assumptions, as in [5] and [46]. Nevertheless, we report them here and in Tab. 1 for completeness. For our computations, we choose an asymmetric specification where preferences are heterogeneous across countries. In particular, the intertemporal elasticity of substitution (IES) of the M countries is evenly spread over the interval $[0.25, 1]$. The welfare weights τ^j need not be specified, as they do not matter for the capital allocation, but only for the consumption allocation which we do not consider. Finally, the parameter A is chosen such that the capital of each country is equal to 1 in the deterministic steady state.

3. Sparse grid interpolation

For the time iteration algorithm we propose in Sec. 4, we need to repeatedly evaluate (policy) functions at arbitrary coordinates within the domain of interest. As a single function evaluation can be very expensive — it involves solving a system of nonlinear equations (cf. (8) and (6)) — we need an efficient interpolation scheme. Our method of choice is adaptive sparse grid interpolation, which we now explain.

Generally speaking, we aim to approximate each individual variable of the policy or value function by a function $f : \Omega \to \mathbb{R}$ as

$$f(\vec{x}) \approx u(\vec{x}) := \sum_i \alpha_i \phi_i(\vec{x}) \tag{11}$$

with coefficients α_i and a set of appropriate (piecewise linear) basis functions $\phi_i(\vec{x})$. Employing standard discretization methods for the high-dimensional domain Ω is out of the question, as ordinary discretization approaches yield too many grid points where

Table 1: Choice of parameters for the IRBC model

Parameter	Symbol	Value
Discount factor	β	0.99
IES of country j	γ^j	a+(j-1)(b-a)/(M-1) with a=0.25, b=1
Capital share	κ	0.36
Depreciation	δ	0.01
Standard deviation of log-productivity shocks	σ	0.01
Autocorrelation of log-productivity	ρ	0.95
Intensity of capital adjustment costs	ϕ	0.50
Aggregate productivity	A	$(1-\beta(1-\delta))/(\alpha\cdot\beta)$
Welfare weights	τ^j	A^{1/γ^j}
Number of countries	M	2,3,4

the functions have to be evaluated. Starting with a one-dimensional discretization scheme that employs N grid points, a straightforward extension to d dimensions by a tensor product construction would lead to N^d grid points, encountering the so-called *curse of dimensionality* [13]. The exponential dependence of the overall computational effort on the number of dimensions is a prohibitive obstacle for the numerical treatment of high-dimensional problems. Sparse grids, on the other hand, are able to alleviate this *curse of dimensionality* by reducing the number of grid points by orders of magnitude, yet with only slightly deteriorated accuracy if the underlying function is sufficiently smooth [8].

In this section, we therefore first provide a brief introduction to classical, i.e., non-adaptive sparse grid interpolation. Subsequently, we also show how the hierarchical structure of the basis functions and the associated sparse grid can be used to refine the grid such that it can better capture the local behavior of the functions to be interpolated. In Sec. 4, we will see in the case of an economic model that adaptive sparse grids outperform classical sparse grids by far when it comes to interpolating functions that exhibit steep gradients or nondifferentiabilities.

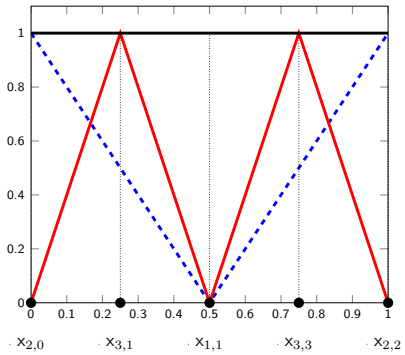

Figure 1: Hierarchical basis functions of V_3 in (17) in one dimension. Level $l = 1$ (solid black), $l = 2$ (dashed blue), and $l = 3$ (solid red).

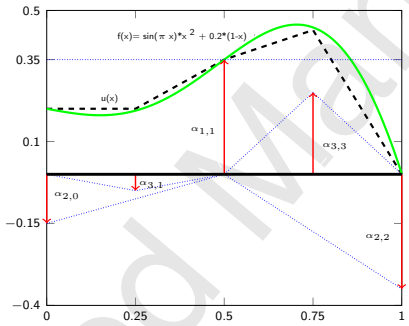

Figure 2: Construction of $u(x)$ interpolating $f(x) = \sin(\pi \cdot x) \cdot x^2 + 0.2 \cdot (1 - x)$ with hierarchical linear basis functions of levels 1, 2, and 3. The hierarchical surpluses $\alpha_{l,i}$ that belong to the respective basis functions are indicated by arrows. They are simply the difference between the function values at the current and the previous interpolation levels.

3.1. Notation

Following [8] and [17], we first introduce some notation and definitions that we will require later on. For all our considerations, we will focus on the domain $\Omega = [0,1]^d$, where d is the dimensionality of the economic problem. This situation can be achieved for other domains by a proper rescaling. Moreover, let $\vec{l} = (l_1, ..., l_d) \in \mathbb{N}^d$ and $\vec{i} = (i_1, ..., i_d) \in \mathbb{N}^d$ denote multi-indices, and define $|\vec{l}|_1 := \sum_{t=1}^{d} l_t$ and $|\vec{l}|_\infty := \max_{1 \le t \le d} l_t$.

3.2. Hierarchical basis functions

We use a sparse grid interpolation method that is based on a hierarchical decomposition of the underlying approximation space. Such a hierarchical structure is con-

venient both for local adaptivity (see Sec. 3.4) and for the use of massively parallel architectures (see Sec. 4). We now explain this hierarchical structure, starting with the one-dimensional case, i.e., $\Omega = [0,1]$. Afterwards, we will extend it to the multivariate case using tensor products. The equidistant sparse grid interpolant we use below [8, 17] consists of a combination of nested one-dimensional grids of different refinement levels. For a given level $l \in \mathbb{N}$, the grid points on $[0,1]$ are distributed as

$$x_{l,i} = \begin{cases} 0.5, & l = i = 1, \\ i \cdot 2^{1-l}, & i = 0, ..., 2^{l-1},\ l > 1. \end{cases} \tag{12}$$

The corresponding piecewise linear basis functions for $x \in [0,1]$ are given by

$$\begin{aligned} &\phi_{l,i}(x) \\ &= \begin{cases} 1, & l = i = 1, \\ \max(1 - 2^{l-1} \cdot \left|x - x_{l,i}\right|, 0) & i = 0, ..., 2^{l-1},\ l > 1. \end{cases} \end{aligned} \tag{13}$$

Note that the basis function of level 1 is a constant rather than a hat function, which is different from many other sparse grid constructions (see, e.g., [8, 17], and references therein). The one-dimensional basis functions can be extended to d-dimensional ones on the unit cube $\Omega = [0,1]^d$ by a tensor product construction. For each grid point $\vec{x}_{\vec{l},\vec{i}} = (x_{l_1,i_1}, ..., x_{l_t,i_t})$, an associated piecewise d-linear basis function $\phi_{\vec{l},\vec{i}}(\vec{x})$ is defined as the product of the one-dimensional basis functions (cf. (14))

$$\phi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{t=1}^{d} \phi_{l_t,i_t}(x_t). \tag{14}$$

Next, the hierarchical increment spaces $W_{\vec{l}}$ are defined by

$$W_{\vec{l}} := \operatorname{span}\{\phi_{\vec{l},\vec{i}} : \vec{i} \in I_{\vec{l}}\} \tag{15}$$

with the index set $I_{\vec{l}}$ given as

$$I_{\vec{l}} := \begin{cases} \{\vec{i} : i_t = 1, 1 \le t \le d\} & \text{if } l = 1, \\ \{\vec{i} : 0 \le i_t \le 2,\ i_t \text{ even}, 1 \le t \le d\} & \text{if } l = 2, \\ \{\vec{i} : 0 \le i_t \le 2^{l_t - 1},\ i_t \text{ odd}, 1 \le t \le d\} & \text{else.} \end{cases} \tag{16}$$

Fig. 1 depicts the first three levels of the associated $1d$ hierarchical, piecewise linear

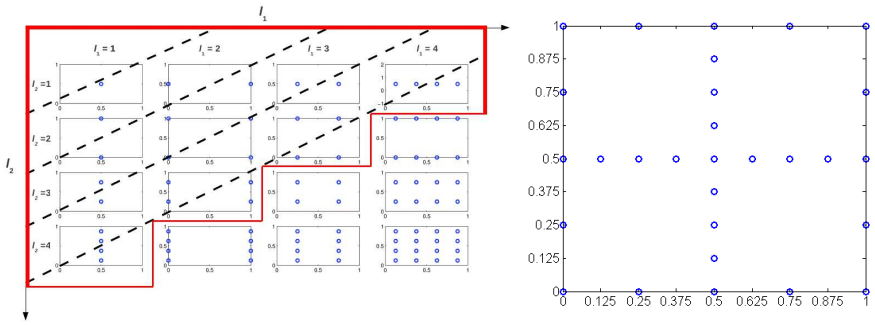

Figure 3: Left: schematic construction of a level 4 sparse grid V_4^S [48] in two dimensions. Right: sparse grid space V_4^S in 2 dimensions, constructed according to (22) from the increments displayed in the left graphic.

basis functions. Consequently, the hierarchical increment spaces $W_{\vec{l}}$ are related to the space V_n of $d-$*linear* functions with mesh size $h_n = 2^{1-n}$ in each dimension by

$$V_n := \bigoplus_{l_1=1}^{n} \cdots \bigoplus_{l_d=1}^{n} W_{\vec{l}} = \bigoplus_{|l|_\infty \leq n} W_{\vec{l}}, \tag{17}$$

leading to a full grid with $\mathcal{O}(2^{nd})$ grid points. The interpolant of f, namely, $u(\vec{x}) \in V_n$, can uniquely be represented by

$$f(\vec{x}) \approx u(\vec{x}) = \sum_{|l|_\infty \leq n} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot \phi_{\vec{l},\vec{i}}(\vec{x}) \tag{18}$$

with $\alpha_{\vec{l},\vec{i}} \in \mathbb{R}$. Note that the coefficients $\alpha_{\vec{l},\vec{i}} \in \mathbb{R}$ are commonly termed the hierarchical surpluses [24, 8]. They are simply the difference between the function values at the current and the previous interpolation levels (see Fig. 2). As we have chosen our set of grid points to be nested, i.e., such that the set of points X^{l-1} at level $l-1$ with support nodes $\vec{x}_{\vec{l},\vec{i}}$ is contained in X^l, namely, $X^{l-1} \subset X^l$, the extension of the interpolation level from level $l-1$ to l only requires us to evaluate the function at grid points that are unique to X^l, that is, at $X^l_\Delta = X^l \backslash X^{l-1}$. For a sufficiently smooth function f (which we will make precise in the next section) and its interpolant $u \in V_n$ [8], we obtain an asymptotic error decay of

$$\| f(\vec{x}) - u(\vec{x}) \|_{L_2} \in \mathcal{O}\left(h_n^2\right), \tag{19}$$

but at the cost of

$$\mathscr{O}\left(h_n^{-d}\right) = \mathscr{O}\left(2^{nd}\right) \tag{20}$$

function evaluations, encountering the *curse of dimensionality*.

3.3. Ordinary sparse grids

As a consequence of the *curse of dimensionality*, the question that needs to be answered is how we can construct discrete approximation spaces that are better than V_n in the sense that the same number of invested grid points leads to a higher order of accuracy. The classical sparse grid construction arises from a cost-to-benefit analysis (see, e.g., [8, 17, 24], and references therein) in function approximation. Thereby, functions $f(\vec{x}) : \Omega \to \mathbb{R}$ which have bounded second mixed derivatives are considered. For such functions, the hierarchical coefficients $\alpha_{\vec{l},\vec{i}}$ (see (18) and [8]) rapidly decay, namely,

$$|\alpha_{\vec{l},\vec{i}}| = \mathscr{O}\left(2^{-2|\vec{l}|_1}\right). \tag{21}$$

The strategy for constructing a sparse grid thus is to leave out those subspaces among the full grid space V_n that only contribute little to the interpolant [8]. An optimization with respect to the number of degrees of freedom, i.e., the grid points, and the resulting approximation accuracy directly lead to the sparse grid space V_n^S of level n, defined by

$$V_n^S := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}. \tag{22}$$

In Fig. 3, we depict its construction for $n = 4$ in two dimensions. V_4^S consists of the hierarchical increment spaces $W_{(l_1,l_2)}$ for $1 \leq l_1, l_2 \leq n = 4$. The area enclosed by the red bold lines marks the region where $|\vec{l}| \leq n + d - 1$, fulfilling (16). The blue dots represent the grid points of the respective subspaces. Finally, the dashed black lines indicate the hierarchical increment spaces for constant $|\vec{l}|$. Note that the sparse grid contains only 29 support nodes, whereas a full grid would consist of 81 points.

The concrete choice of subspaces depends on the norm in which we measure the error. The result obtained in (22) is optimal for the L_2-norm and the L_∞-norm [8]. The number of grid points required by the space V_n^S is now given by [8, 17]

$$|V_n^S| = \mathscr{O}\left(h_n^{-1} \cdot \left(\log(h_n^{-1})\right)^{d-1}\right). \tag{23}$$

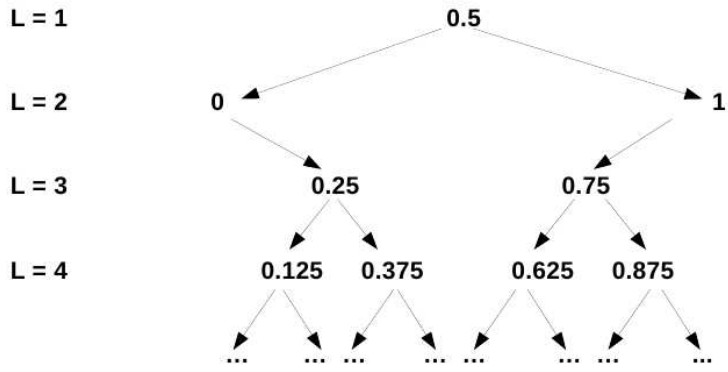

Figure 4: One-dimensional tree-like structure of a classical sparse grid (cf. Sec. 3.3) for the first three hierarchical levels.

This is of order $\mathcal{O}\left(2^n \cdot n^{d-1}\right)$, which is a significant reduction of the number of grid points, and thus of the computational and storage requirements compared to $\mathcal{O}\left(2^{nd}\right)$ of the full grid space $|V_n|$ (see Fig. 3). In analogy to (18), a function $f \in V_n^S \subset V_n$ can now be expanded by

$$f_n^S(\vec{x}) \approx u(\vec{x}) = \sum_{|l|_1 \leq n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot \phi_{\vec{l},\vec{i}}(\vec{x}). \tag{24}$$

The asymptotic accuracy of the interpolant deteriorates only slightly from $\mathcal{O}\left(h_n^2\right)$ in the case of the full grid (cf. (19)) down to

$$\mathcal{O}\left(h_n^2 \cdot \log(h_n^{-1})^{d-1}\right), \tag{25}$$

as shown e.g. in [8, 17]. Taken together, (23) and (25) demonstrate why sparse grids are so well suited for high-dimensional problems. In contrast to full grids, their size increases only moderately with dimension, while the accuracy they provide is nearly as good as the one of full grids.

3.4. Adaptive sparse grids

In many economic applications [5], the functions to be interpolated do not meet the regularity conditions assumed above, but instead have steep gradients, nondifferentiabilities, or even finite discontinuities. In such cases, the classical sparse grid methods

outlined so far may fail to provide a good approximation. One effective way to overcome this problem is to adaptively refine the sparse grid in regions with high function variation and spend fewer points in regions of low variation (see, e.g., [8, 9, 26], and references therein). The working principle of the refinement strategy we use is to monitor the size of the hierarchical surpluses (see (21)), which reflect the local irregularity of the function. For functions with small mixed-derivatives the hierarchical surpluses rapidly converge to zero as the level l tends to infinity (cf. (21)). On the other hand, a nondifferentiability or discontinuity can often be identified by large and slowly decaying hierarchical surpluses. Therefore, we use the hierarchical surpluses as an error indicator and refine the grid around a grid point if its surplus $\alpha_{\vec{l},\vec{i}}$ satisfies

$$|\alpha_{\vec{l},\vec{i}}| \geq \varepsilon, \tag{26}$$

for a so-called refinement threshold $\varepsilon \geq 0$. Technically, the adaptive grid refinement can be built on top of the hierarchical grid structure. The points of the classical sparse grid form a tree-like data structure, as displayed in Fig. 4 for the one-dimensional case. Going from one level to the next, we see that there are two *sons* for each grid point (if $l \neq 2$). For example, the point 0.5 from level $l = 1$ is the *father* of the points 0 and 1 from level $l = 2$. In the d-dimensional case, there are consequently two *sons* per dimension for each grid point, i.e., $2d$ *sons* in total. Whenever the criterion given by (26) is satisfied, these $2d$ neighbor points of the current point are added to the sparse grid.[6] In this way, we can adapt to nondifferentiabilities induced by occasionally binding constraints that are common in economic models. While existing methods can adapt very precisely to these nondifferentiabilities [49, 27], adaptive sparse grids work in much higher dimensions.

[6] We point out that in our application in Sec. 4 we interpolate several policies on one grid, i.e., we interpolate a function $f : \Omega \to \mathbb{R}^m$. Therefore, we get m surpluses at each grid point and we thus have to replace the refinement criterion in (26) by $g\left(\alpha^1_{\vec{l},\vec{i}}, \ldots, \alpha^m_{\vec{l},\vec{i}}\right) \geq \varepsilon$, where the refinement choice is governed by a function $g : \mathbb{R}^m \to \mathbb{R}$. A natural choice for g is the maximum function, which we will use in Sec. 4.

4. Scalable sparse grid time iteration algorithm

We now describe how to solve the IRBC model introduced in Sec. 2.2 using adaptive sparse grids as presented in Sec. 3. For this purpose, we build a time iteration algorithm (see, e.g., [6]) that uses adaptive sparse grid interpolation in each iteration step (cf. Sec. 4.1). We parallelize this algorithm by a hybrid parallelization scheme using MPI [38], Thread Building Blocks [10], and CUDA/Thrust [11], as outlined in Sec. 4.2

4.1. Time iteration

The time iteration algorithm that we use to compute a policy function satisfying (2) is based on the following heuristic: Solve the equilibrium conditions of the model for today's policy $p : X \to Y$ taking as given an initial guess for the function that represents next period's policy, p_{next}; then, use p to update the guess for p_{next} and iterate the procedure until convergence. Note that in the case of Pareto optimal problems, as the one solved in this paper, convergence of time iteration can be derived from convergence of value function iteration (see [2] for a comprehensive study of value function iteration and its convergence properties) and even explicit error bounds for the approximate policy functions can be obtained under strong concavity (see [50]). For non-optimal economies, results about convergence of time iteration and also the existence of recursive equilibria are harder to obtain, yet are available for large classes of models with heterogeneous agents, incomplete markets, externalities, discretionary taxation and other salient features of applied models (see, e.g., [51, 52]).

The structure of our time iteration algorithm is given in Algorithm 1. [7] Two remarks about the maximal refinement level, L_{max}, are in place. First, the classical sparse grid of level L is obtained as a special case of this algorithm by setting $L_{max} = L_0 = L$.

[7] Note that the formal iterative structure of Algorithm 1 is similar to the approach taken by [16] to solve the HJB equation. However, there are a couple of differences worth mentioning. First, since we aim to iteratively solve for a multivariate policy function, our adaptive refinement criterion had, as pointed out earlier, to be extended to the multivariate case, whereas [16] consider a single-valued function approximation. On the other hand, [16] allow for adaptive coarsening when iterating from one time step to the next, while we only allow for adaptive refinement.

Data: Initial guess p_{next} for next period's policy function. Approximation accuracy $\bar{\eta}$. Maximal refinement level L_{max}. Starting refinement level $L_0 \leq L_{max}$.

Result: The (approximate) equilibrium policy function p.

while $\eta > \bar{\eta}$ **do**

  Set $l = 1$, set $G \subset X$ to be the level 1 grid on X, and set $G_{old} = \emptyset, G_{new} = \emptyset$.

  while $G \neq G_{old}$ *and* $l \neq L_{max}$ **do**

    for $g \in G \setminus G_{old}$ **do**

      Compute the optimal policies $p(g)$ by solving the system of equilibrium conditions

$$0 = \mathbb{E}\left\{ f\Big(g, x_{t+1}, p(g), p_{next}(x_{t+1})\Big) | g, p(g) \right\},$$
$$x_{t+1} \sim F(\cdot | g, p(g)),$$

      given next period's policy p_{next}.

      Define the policy $\tilde{p}(g)$ by interpolating $\{p(g)\}_{g \in G_{old}}$.

      if $l < L_{max}$ *and* $\|p(g) - \tilde{p}(g)\|_\infty > \varepsilon$, **then**

        Add the neighboring points (*sons*) of g to G_{new}.

      end

    end

    Set $G_{old} = G$, set $G = G_{old} \cup G_{new}$, set $G_{new} = \emptyset$, and set $l = l + 1$.

  end

  Calculate (an approximation for) the error: $\eta = \|p - p_{next}\|_\infty$.

end

Algorithm 1: Overview of the crucial steps of the time iteration algorithm.

Second, for the adaptive sparse grid, one could in principle set the maximum refinement level L_{max} to a very large value such that it is never reached for a given refinement threshold. However, this can create practical problems: in case of high curvature or non-differentiabilities, the hierarchical surpluses may decrease very slowly and the algorithm may not stop to refine until a very high interpolation level. Thus, as one has no reasonable upper bound for the number of grid points created by the refinement

procedure, we have to set a maximum refinement level.

An important detail of the implementation is the integration procedure used to evaluate the expectations operator. In case of the IRBC application, the expectation term in Equ. (8) has to be evaluated by integrating over the normally distributed productivity shocks. As we want to focus on the grid structure, we chose an integration rule that is simple and fast. In particular, we use a simple monomial rule that exploits the normality assumption and uses just two evaluation points per shock, thus $2(M+1)$ points in total (see, [6], with references therein). As we apply the same rule along the time iteration algorithm as well as for the error evaluation, this choice factors out the question of finding integration procedures that are both accurate and efficient. In principle integration could also be carried out using an (adaptive) sparse grid [25, 9], yet not over the same space that the policy functions are interpolated on. Therefore, we view integration as a problem that is orthogonal to the choice of the grid structure, and thus do not focus on it.

4.2. Hybrid parallelization scheme

In each step of the above time iteration procedure the updated policy function is determined using a hybrid-parallel algorithm, as shown in Fig. 5. We construct adaptive sparse grids by distributing the newly generated grid points via MPI within a refinement step among multiple, multithreaded processes. The points that are send to one particular compute node are then further distributed among different threads. Each thread then solves a set of nonlinear equations for every single grid point assigned to it. The set of nonlinear equations—given by (8)–(10) in our application—is solved with `Ipopt` [53], which is a high-quality open-source software for solving nonlinear programs (`http://www.coin-or.org/Ipopt/`). On top of this, we add an additional level of parallelism. When searching for the solution to the equation system at a given point, the algorithm has to frequently interpolate the function computed in the previous iteration step. These interpolations take up 99% of the computation time needed to solve the equation system. As they have a high arithmetic intensity—that is to say, many arithmetic operations are performed for each byte of memory transfer and access—they are perfectly suited for GPUs. We therefore offload parts of the inter-

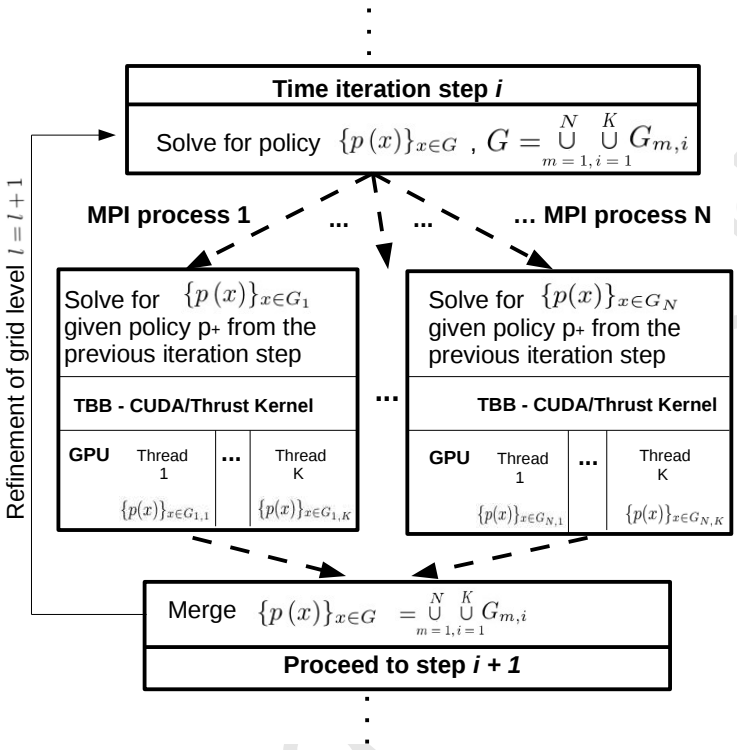

Figure 5: Schematic representation of the hybrid parallelization of a time iteration step presented in Algorithm 1 and Sec. 4.2. Each MPI process is using TBB and a CUDA/Thrust kernel for the function evaluation.

polation from the compute nodes to their attached accelerators (cf. Sec. 4.3 for more details). Hence, CPU cores and the GPU device of a single node are utilized through multiple threads, and MPI is used for internode communication only.

Given the limited availability of unified multicore CPU/GPU programming models, such as OpenMP 4, and our aim to perform more aggressive manual optimizations, we decided to develop two separate code paths: GPU kernels are implemented with Thrust [11], while CPU multithreading and CPU/GPU workload partitioning is organized with Thread Building Blocks [10].

4.3. Single node optimization and parallelization

In order to solve the IRBC model in minimal time, we aim to utilize the computational resources available on each compute node in an efficient manner. Targeting

primarily hybrid CPU+GPU compute nodes, our general strategy is to map the homogeneous workload onto a combination of CPU threads and GPU kernels. In this section, we explain the steps taken during IRBC code optimization. The resulting gains are then reported in Sec. 5.1.

Explicit programming of mathematical notations "as is" is an essential starting point for any scientific application. The optimizations the compiler is able to perform are, however, not always of the same quality as manual math expressions folding. The following basic source transformations increased the odds of getting a reasonably efficient binary code:

- eliminate floating-point divisions
- eliminate redundant branching
- eliminate redundant computations, conserving the memory throughput.

Above's code changes often work in combination. For instance, precomputing directly usable index arrays made it possible to eliminate branching in the section of the code that computes the linear basis functions. The GPU version of the policy function evaluation is implemented with Thrust's `transform_reduce`. Arrays of read-only indices are transposed to fitful coalescing requirements. One GPU thread handles 4 consecutive indices that are loaded with a single `int4` (`LD.128`) memory transaction.

Multithreading on a single node CPU is implemented with Thread Building Blocks (TBB). Moreover, one of the TBB-managed threads is exclusively used for the GPU kernels dispatch. CPU and GPU threads leverage TBB's automatic workload balancing based on stealing tasks from slower workers (see Fig. 6). Our code performs floating-point computations in double precision. Modern SIMD CPUs are able to handle 4 double values in a single instruction using 32-byte AVX vector registers (see Fig. 7). The use of AVX not only increases effective arithmetic throughput in compute-bound applications, but also results in a better register allocation and reduced cache pressure in memory-bound applications. Therefore, vectorization is preferred, regardless the class of application. Scalar-vector code transition is mostly straight-forward; however, special attention should be paid to element-wise accesses within AVX vectors: they are

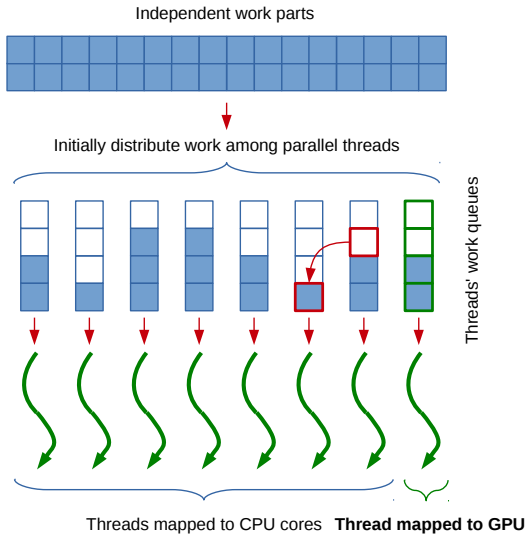

Figure 6: Hybrid multithreading with Intel TBB: $(N-1)$ threads on CPU, 1 thread – for GPU; TBB balances workloads automatically using "work stealing"

expensive and should be avoided. In other words, the most efficient vectorization could be achieved if the code is fully vectorized from loading inputs to storing outputs, without being interrupted by scalar regions. Specifically for the AVX version of the policy function evaluation, we had to adapt arithmetics, 0-comparison branch and `abs` function all to handle 4 grid points at once, which resulted into 2.1× overall performance improvement.

Thrust's `transform_reduce` implementation allocates the GPU memory buffer to keep partial sums, which is not exposed to the user. The reuse of this buffer across subsequent reductions with equal parameters is not supported. As result, Thrust accompanies each reduction with an allocation and deallocation of a small memory region. We eliminated the overhead of redundant allocations/deallocations by providing alternative Thrust-aware `cudaMalloc`/`cudaFree` implementations that allocate the requested buffer one time, and then pass the existing allocation to all subsequent requests, without freeing it. This modification reduced the total execution time by approximately 7%, as shown in Fig. 8.

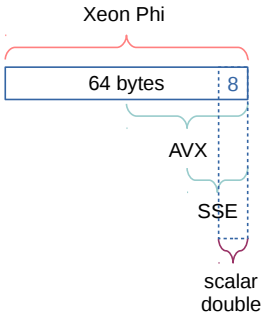

Figure 7: Vector registers on modern CPUs: a scalar program can utilize only 1/4 of computational parallelism on AVX-enabled CPUs, e.g. the SandyBridge.

Most of the read-only data used by the policy function evaluation is shared across all invocations. The only exception is the x coordinate vector, whose size equals to the dimensionality of economic problem, typically a small value. Given that PCI-E data transfer reaches optimal bandwidth for vector sizes of at least several megabytes, x-vector copying always has low efficiency. One simple method to eliminate this small inefficient vector `cudaMemcpy` is to append the vector elements directly to the kernel argument list (as scalars):

```
kernel<<<grid, block>>>(..., x[0], x[1], ..., x[DIM - 1]);
```

Scalar elements are then assembled back into the local array in order to keep simple indexing. This technique requires that we hard-coded the dimensionality of the economic problem into the kernel source. Knowing its value, the compiler will very likely perform complete unrolling of the corresponding loop both in the CPU and GPU versions, resulting in less branching. The local array will be mapped onto registers. Our implementation deploys JIT-compilation to dynamically compile CPU/GPU kernels, hard-coding the required dimension value, which could be scripted in a number of ways. Our implementation uses C macros for the x-vector manipulations, invokes the compiler from the running program and loads the compiled object as a shared library (`dlopen/dlsym`). This saves on the time of separate host-device memory transfers and leads to a speedup of about 15%. On the downside, the hard-coded kernels must be generated during program runtime, inducing some overhead from the compilation and

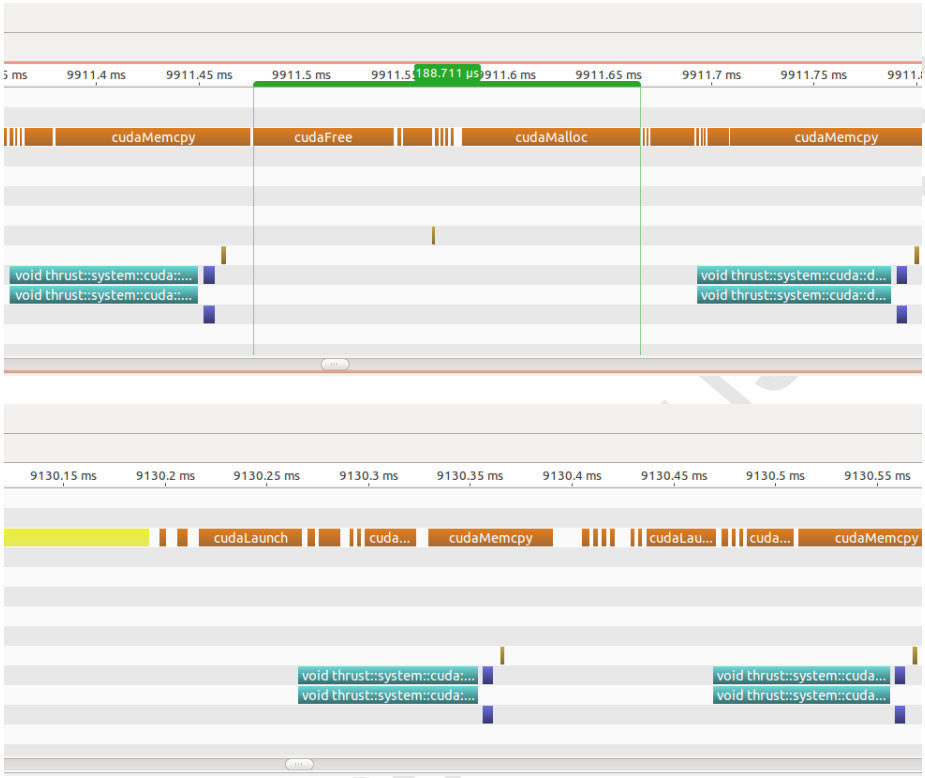

Figure 8: Eliminating Thurst's GPU scratch space memory allocation overhead: original implementation profile (top), reusing single-time allocation across all Thrust kernel calls (bottom). The gap between kernel launches is approximately 2 times smaller, thanks to elimination of `cudaMalloc`/`cudaFree` (green range).

disk I/O. The quantitative impact of all optimizations discussed here are summarized in Sec. 5.1 and Figure 9.

5. Numerical experiments

For our scaling experiments, we consider an 8-dimensional economic problem with four countries in Sec. 5.1 and 5.2, and four to 8-dimensional models in Sec. 5.3. [8] In

[8]Note that in computational economics, there are no standard baseline tests such as, for instance, the Sedov-Taylor blast wave test in physics [54]. What comes closest to being a standard high-dimensional test case is the the IRBC model used in [46]. However, to demonstrate the potential of adaptive grids, we include irreversibility constraints that make the model much harder to solve (cf., Sec. 2.2). Solving dynamic

this section, we report on the single node performance achieved by the various code optimizations described in Sec. 4.3. Moreover, we evaluate the strong scaling efficiencies of the IRBC model on the new Cray XC30 "Piz Daint" system. Finally, we discuss solutions to the IRBC models and show how adaptive sparse grids can speed up the computations.

We deploy the IRBC model on the $5,272$-node Cray XC30 "Piz Daint" system installed at Swiss National Supercomputing Centre (CSCS). Cray XC30 compute nodes combine 8-core Intel Xeon E5-2670 (SandyBridge) CPUs with $1\times$ NVIDIA Tesla K20X GPU. The IRBC model is compiled with GNU compilers and CUDA Toolkit 5.0.

5.1. Single node performance

To give a measure of how the various optimizations discussed in Sec. 4.3 impact the performance of the time iteration code on a single CPU thread, GPU and entire node, we evaluated the second refinement levels of a single time step from the IRBC model outlined before. This instance consists of 128 grid points, $1,152$ variables and constraints. The results are summarized in Fig. 9. and indicate a total speedup of about $30\times$ when going from the naive single CPU thread implementation to a more efficient version utilizing both CPU and GPU resources of the entire node. Most notably, we can see that a single-threaded GPU is about $12\times$ faster than a single-threaded CPU, leading to an overall speedup of $\sim$50% when the entire node is utilized in a multi-threaded mode[9] (see Fig. 9).

economic models with such constraints in high dimensions was not possible before and there is thus no baseline to compare to.

[9]Note that [32] observed in their examples speedups one to two orders of magnitude larger than the one observed here when invoking GPUs for function evaluations on sparse grids. However, their results are not directly comparable to ours due to four reasons: first they compare a so-called "iterative" sparse grid evaluation to a "recursive" one that serves as baseline. We, on the other hand, compare a multi-threaded algorithm to a single-threaded one, where the function evaluation is performed in an "iterative" fashion, similar to [32]. Secondly, their scaling experiments are based on a test case where tens of thousands of function evaluations can be performed at once, whereas we can only perform a couple of hundred function evaluations per GPU invocation. Third, their most significant speedups were observed in single presicion

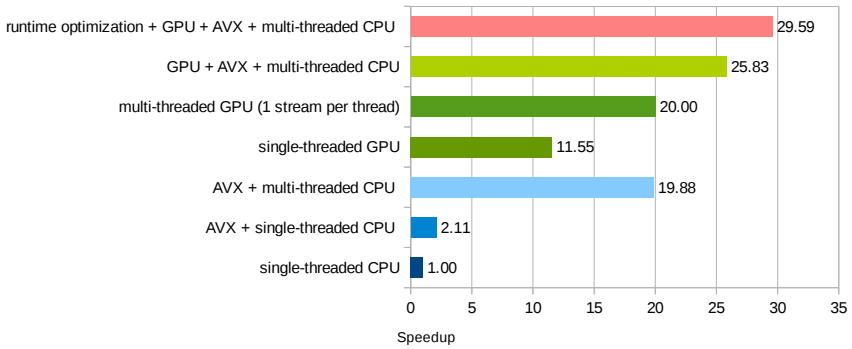

Figure 9: Comparison of walltimes for different IRBC model code variants on a single node of Piz Daint (speedup against scalar serial version). Benchmarked configuration: 8-dimensional model with 128 grid points and $1,152$ variables and constraints.

5.2. Strong Scaling

We now report strong scaling efficiency of our code. The test problem is again a single timestep of an 8-dimensional economic problem with 4 countries. In order to provide a consistent benchmark, we used a nonadaptive classical sparse grid of refinement level 6. This instance has a total of $510,633$ variables and constraints per time step [10]. The economic test case was solved with increasingly larger numbers of nodes (from 1 to $2,048$ nodes). Fig. 10 shows the execution time and scaling on different levels and their ideal speedups. We used 1 MPI processes per multi-threaded Intel SandyBridge node, of which each offloads part of the function evaluation to the K20x GPU (cf. Sec. 4.2 and Fig.5). For this benchmark, the code scales nicely up to the order of 256 to 512 nodes. Thus, combined with the speedup gains due to TBB, the GPU and the code optimizations reported in Secs. 4.3 and 5.1, we attain an overall speedup of more than three orders of magnitude for our benchmark. The dominant limitation to the strong scaling stems from the fact that within the first few refinement levels, the ratio of "points to be evaluated to MPI processes" is often smaller than one

computations, whereas we have to run in double-precision mode. Finally, [32] were using different hardware where for instance GPUs were attached to one compute node.

[10] The sparse grid under considerartion consists of 56,737 points that each hold 9 variables.

with increasing node numbers, i.e., there are MPI processes idling, as can be seen in Fig. 10. Moreover, the workload sometimes may be unbalanced in the case of large node numbers in a sense that, e.g. one MPI process gets 2 points to work on, while a second one obtains only 1 point to work on. The better parallel efficiency on the higher refinement levels is due to the fact we have many more points available on this refinement level, so the workload is somewhat fairer distributed among the different MPI processes. Thus, strong scaling efficiencies will be much better for higher-dimensional models ($d > 8$), as the number of newly generated grid points grows faster with increasing refinement levels. In a 24-dimensional model, for example, the points of a fixed sparse grid grow with the refinement level by 48, 1,152, 18,496, and 224,304 points, i.e. the ratio of grid points to be evaluated in the individual refinement levels per MPI process is considerably larger compared to our example.

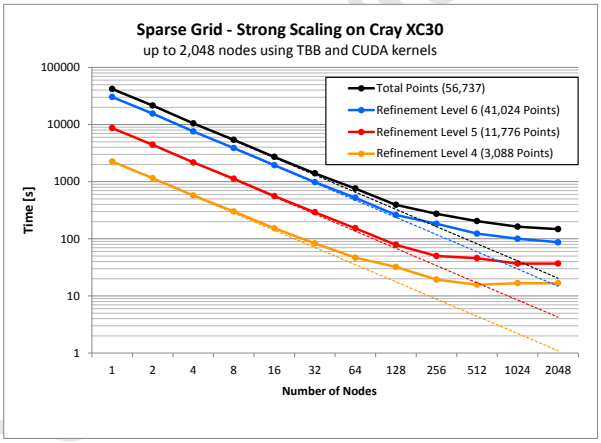

Figure 10: Strong scaling plots on Piz Daint for an IRBC model using 6 levels of grid refinements and in total 56,737 points. "Total" shows the entire simulation time up 2,048 nodes. We also show execution times for the computational subcomponents on different refinement levels, e.g, for refinement level 6 using 41,024 points, refinement level 5 using 11,776 points, and refinement level 4 using 3,088 points. Dotted lines show ideal speedups.

5.3. Convergence of time iteration

In order to gain an understanding of how the adaptivity in our algorithm can speed up computations in nonsmooth economic problems, we compare adaptive and non-

adaptive solutions of the IRBC model with binding constraints outlined in Sec. 2. For this purpose, we define the L_∞- and the L_2-error as (cf. Algorithm 1)

$$L_\infty = \max_{i=1,\dots,\Theta} |p(x_i) - p_{next}(x_i)|, \tag{27}$$

and

$$L_2 = \frac{1}{\Theta}\left(\sum_{i=1}^{\Theta} (p(x_i) - p_{next}(x_i))^2\right)^{\frac{1}{2}}, \tag{28}$$

i.e., we interpolate the two consecutive policy functions p and p_{next} at $\Theta = 10,000$ test points that were randomly generated from a uniform distribution over the state space. In Fig. 11, we compare the decaying L_2 and L_∞- error for a complete simulation of an 8-dimensional model, once run with a fixed sparse grid of level 7 (refinement level $L_{max} = 6$), and once run with an adaptive sparse grid of a refinement threshold $\varepsilon = 0.01$ and a maximum refinement level of six. [11] It is apparent from Fig. 11 that convergence of the time iteration algorithm is rather slow. This is to be expected, as time iteration has, at best, a linear convergence rate.[12] Fig. 11 also shows that adaptive sparse grids are much more efficient in reducing the approximation errors in our model, as they put additional resolution where needed, while not wasting resources in areas of smooth variation. The adaptivity in this particular benchmark reduces the size of the grid by more than one order of magnitude compared to a classical sparse grid (see Tab. 2). Since the interpolation time on sparse grids grows faster than linearly with the number of points (see, e.g., [21]), the walltime is in this experiment reduced by approximately two orders of magnitude. Hence, adaptive sparse grids introduce an additional layer of sparsity on top of the a priori sparse grid structure of the classical sparse grid. In Fig. 12, we illustrate this by displaying 2-dimensional projections of a fixed and an adaptive sparse grid. Thus, we are able to locally mimic an interpolant that is of very high order where needed, while in other regions, only a few points are invested. This

[11] Note that $\varepsilon = 0.01$ and $L_{max} = 6$ in this example were chosen such that the simulations satisfy the order of accuracy desired.

[12] Assuming strong concavity of the return function (in either the state or the choice), [50] show that the policy function of a stochastic dynamic programming problem is Hoelder continuous in the value function and that its convergence rate is the square root of the discount factor.

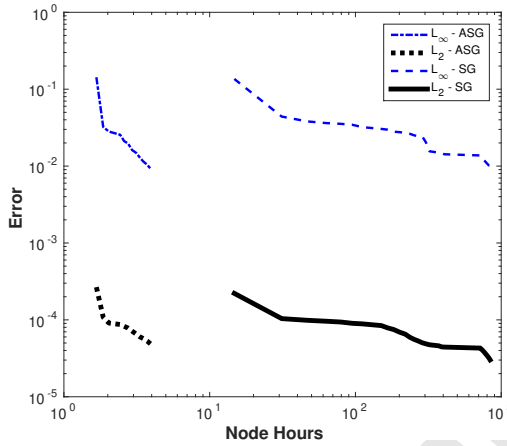

Figure 11: Comparison of the decreasing maximum (L_∞) and L_2-error for conventional (SG) and adaptive sparse grid (ASG) solutions to the $2N = 8$-dimensional nonsmooth IRBC model as a function of node hours on a Cray XC30. We compute these errors for ten thousand points drawn from a uniform distribution over the state space.

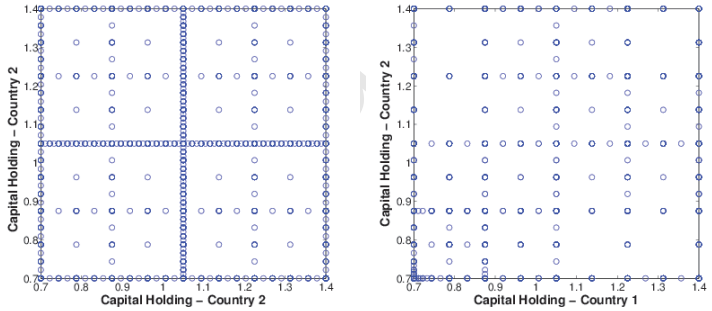

Figure 12: This figure displays 2-dimensional projections of two different grids. The left one is from a classical sparse grid, the right one from an adaptive grid of comparable accuracy. Both grids were generated in the course of running a $2M = 4$-dimensional simulation. The x-axis shows capital holding of country 1, the y-axis shows capital holding of country 2, while the two other axis of the four dimensional grid (productivities of the two countries) are kept fixed at their mean values.

is contrasted by non-adaptive methods which can only provide one resolution over the whole domain. This feature is illustrated in Tab. 2, where we compare the number of grid points for different grid types and dimensions. With adaptive sparse grids, we spend at least one order of magnitude fewer points compared to ordinary sparse grids

Table 2: Average Euler errors for an adaptive sparse grid solution of the nonsmooth IRBC model with increasing dimension. For all dimensions, we use a refinement threshold $\varepsilon = 0.01$ to further refine the grid up to a maximum refinement level of six. The number of required grid points of the adaptive sparse grid solution ($|V_7^{AS}|$) is contrasted by the corresponding grid size of the classical sparse grid ($|V_7^{S}|$) and a full tensor product grid ($|V_7|$). All Euler errors are reported in $\log_{10}$-scale.

Dimension d	Full grid $\lvert V_7\rvert$	$\lvert V_7^S\rvert$	$\lvert V_7^{AS}\rvert$	Euler error
4	17,850,625	2,929	512	-2.88
6	75,418,890,625	15,121	1,679	-2.73
8	318,644,812,890,625	56,737	4,747	-2.66

in order to reach the same accuracy of the interpolant.

Let us now turn our attention to the economic interpretation of our global solutions to the nonsmooth IRBC models. While the L_∞ and L_2 errors displayed in Fig. 11 give an indication about the rate of convergence, we are still lacking a measure of how accurate these solutions are.

To give an economic interpretation to the accuracy of the computed solutions, recall that the policy functions have to satisfy a set of equilibrium conditions. Therefore, it is common practice in economics (see, e.g., [45]) to compute (unit-free) errors in the $M+1$ equilibrium conditions. As in our model, these conditions often mainly consist of *Euler equations*[13], the respective errors are therefore called *Euler errors*. In our IRBC model, there is one Euler equation error for each country $j \in \{1,...,M\}$:

$$\left[\beta \mathbb{E}_t \left\{ \lambda_{t+1} \left[a_{t+1}^j A \kappa (k_{t+1}^j)^{\kappa-1} + (1-\delta) + \frac{\phi}{2} g_{t+2}^j \left(g_{t+2}^j + 2 \right) \right] - \mu_{t+1}^j (1-\delta) \right\} \right] \cdot \left[\lambda_t \left(1 + \phi g_{t+1}^j \right) \right]^{-1} - 1. \quad (29)$$

[13] In economics, the term *Euler equation* has a specific meaning different from its meaning in fluid dynamics. Here, Euler equations are first-order optimality conditions in dynamic equilibrium models. These (difference or differential) equations thus characterize the evolution of economic variables along an equilibrium path.

In addition, there is one additional error from the aggregate resource constraint:

$$\sum_{j=1}^{M}\left(a_t^j\cdot A\cdot(k_t^j)^{\kappa}+k_t^j\cdot\left((1-\delta)-\frac{\phi}{2}\cdot(g_{t+1}^j)^2\right)-k_{t+1}^j-\left(\frac{\lambda_t}{\tau_j}\right)^{-\gamma^j}\right)\cdot\left(\sum_{j=1}^{M}\left(a_t^j\cdot A\cdot(k_t^j)^{\kappa}+k_t^j\cdot\left(-\frac{\phi}{2}\cdot(g_{t+1}^j)^2\right)\right)\right)^{-1}. \quad (30)$$

In case of the IRBC model with irreversible investment there is one additional complication. Denoting the error defined in Eq. 29 by EE^j and defining the percentage violation of the irreversibility constraint by

$$IC^j \equiv 1-\frac{k_{t+1}^j}{k_t^j\cdot(1-\delta)} \quad (31)$$

the error is now given by

$$\max\left(EE^j, IC^j, \min\left(-EE^j, -IC^j\right)\right). \quad (32)$$

The economic reason for this functional form of the error is that the optimal level of investment might be negative and thus not feasible due to the irreversibility constraint. In this case, the violation or slackness in the constraint (that is, IC^j or $-IC^j$) has to be taken into account when calculating the economic error (see [5] for a more detailed explanation of (32)). To calculate the $M+1$ errors at a given point in the state space, we evaluate the terms in (29) to (32) using the computed equilibrium policy function for calculating both today's policy and next period's policy. To generate the statistics on Euler errors reported below we then proceed as follows. We compute the $M+1$ errors for all points in the state space that are visited for ten thousand points drawn from a uniform distribution over the state space. We then take the maximum over the absolute value of these errors, which results in one error for each point. Finally, we compute the average over all points and report the result in $\log_{10}$-scale.

Tab. 2 reports the average Euler errors for adaptive sparse grids of a fixed refinement threshold $\varepsilon = 0.01$, a maximum refinement level $L_{max} = 6$, and increasing dimensionality. We find that the accuracy moderately depends on the dimension of the model. There seems to be a downward trend in the average Euler error. However, this behavior is not surprising. One has to keep in mind that kinks that appear in our

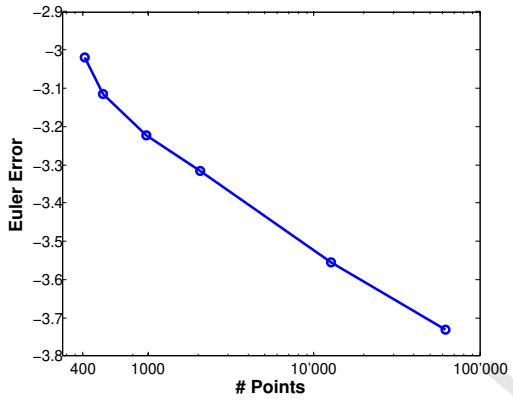

Figure 13: Comparison of the average Euler error (in $\log_{10}$-scale) for adaptive sparse grid solutions to the $2M = 4$-dimensional nonsmooth IRBC model as a function of the number of gridpoints resulting from varying the refinement threshold $\varepsilon = \{0.02, 0.01, 0.005, 0.0025, 0.001, 0.0005\}$.

$2M$-dimensional models are in fact $(2M-1)$-dimensional hypersurfaces. Thus, such objects become much harder to approximate as M increases. Moreover, the maximum refinement $L_{max} = 6$ is binding for dimensions six and eight, while it is not reached for dimension four. Therefore, with a larger L_{max} the errors for higher dimensions would slightly improve relative to the four-dimensional case. More importantly, the Euler errors can be improved substantially by lowering the refinement threshold ε (and increasing the maximum refinement level $L_{max} = 6$)

To show how powerful the adaptive grids are in reducing Euler errors we focus on the four-dimensional case and set L_{max} such that it is never reached. In Figs 13 and 14, the average Euler errors for adaptive sparse grid solutions of the 4-dimensional nonsmooth IRBC model of different refinement thresholds ε are reported. These figures show that the error converges roughly linearly with respect to $1/\varepsilon$ and the number of points. The smaller the chosen refinement threshold, the larger the maximum refinement level reached and the larger the number of gridpoints.

To sum up, the hybrid parallel time iteration algorithm presented in this paper can successfully compute global solutions of high-dimensional, (nonsmooth) dynamic stochastic economic models of a level of complexity not possible before—models with occasionally binding constraints have so far been tractable only in low-dimensional

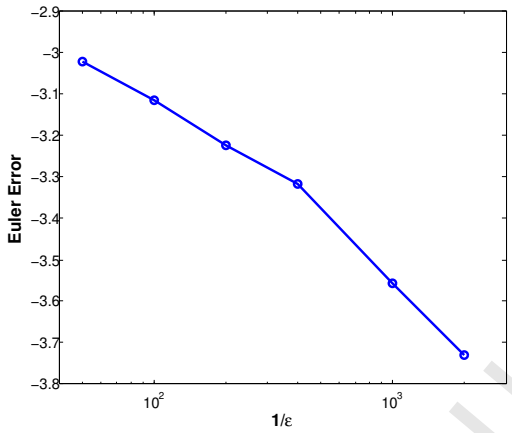

Figure 14: Average Euler error (in $\log_{10}$-scale) for adaptive sparse grid solutions to the $2M = 4$-dimensional nonsmooth IRBC model as a function of the inverse of the refinement threshold $1/\varepsilon$.

cases [27, 28, 29].

6. Conclusion

Solving complex high-dimensional dynamic stochastic economic models numerically in reasonable time—i.e., in hours or days—imposes a variety of problems. In this work, we developed an effective strategy to address these challenges by combining adaptive sparse grids, time iteration methods, and high-performance computing in a powerful toolkit that can handle a broad class of models up to a level of heterogeneity not seen before.

First, using (adaptive) sparse grids alleviates the *curse of dimensionality* imposed by the heterogeneity of the economic models. Second, they can successfully resolve non-smooth policy functions, as they put additional resolution where needed, while not wasting resources in areas of smooth variation. High-performance computing on the other hand enters the picture when we aim to minimize the time-to-solution. By exploiting the generic structure common to many dynamic economic models, we implemented a hybrid parallelization scheme that uses state-of-the-art parallel computing paradigms. It minimizes MPI interprocess communication by using TBB and partially offloads the function evaluations to GPUs.

Numerical experiments on Piz Daint (Cray XC30) at CSCS show that our code is very scalable and flexible. In the case of our intermediate-sized, 8-dimensional IRBC benchmark, we found very good strong scaling properties up to the order of 256 to 512 nodes. The dominant limitation to the strong scaling stems from the fact that within the first few refinement levels, the ratio of "points to be evaluated to MPI processes" is often smaller than 1 with increasing node numbers, i.e. there are MPI processes idling for some time. This all suggests that our framework is very well suited for large-scale economic simulations on massively parallel high-performance computing architectures.

Acknowledgments

The authors would like to thank Felix Kübler (University of Zurich) for helpful discussions and support. Computing time on "Piz Daint" at the Swiss National Supercomputing Center was provided by a USI-CSCS allocation contract. Additionally, this work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID s555.

References

References

[1] L. Ljungqvist, T. J. Sargent, Recursive macroeconomic theory, Mit Press, 2004.

[2] N. Stokey, R. Lucas, E. Prescott, Recursive methods in economic dynamics, Cambridge MA.

[3] D. Krüger, F. Kubler, Computing equilibrium in OLG models with stochastic production, Journal of Economic Dynamics and Control 28 (7) (2004) 1411 – 1436. doi:http://dx.doi.org/10.1016/S0165-1889(03)00111-8.
URL http://www.sciencedirect.com/science/article/pii/S0165188903001118

[4] J. Bengui, E. G. Mendoza, V. Quadrini, Capital mobility and international sharing of cyclical risk, Journal of Monetary Economics 60 (1) (2013) 42–62.
URL http://ideas.repec.org/a/eee/moneco/v60y2013i1p42-62.html

[5] J. Brumm, S. Scheidegger, Using adaptive sparse grids to solve high-dimensional dynamic models, Available at SSRN 2349281.

[6] K. L. Judd, Numerical methods in economics, The MIT press, 1998.

[7] J. P. Rust, Numerical dynamic programming in economics, in: H. M. Amman, D. A. Kendrick, J. Rust (Eds.), Handbook of Computational Economics, 1st Edition, Vol. 1, Elsevier, 1996, Ch. 14, pp. 619–729.
URL http://EconPapers.repec.org/RePEc:eee:hecchp:1-14

[8] H.-J. Bungartz, M. Griebel, Sparse grids, Acta Numerica 13 (2004) 1–123.

[9] X. Ma, N. Zabaras, An adaptive hierarchical sparse grid collocation algorithm for the solution of stochastic differential equations, J. Comput. Phys. 228 (8) (2009) 3084–3113. doi:10.1016/j.jcp.2009.01.006.
URL http://dx.doi.org/10.1016/j.jcp.2009.01.006

[10] J. Reinders, Intel Threading Building Blocks, 1st Edition, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2007.

[11] N. Bell, J. Hoberock, Thrust: A productivity-oriented library for CUDA, GPU Computing Gems 7.

[12] J. J. Dongarra, A. J. van der Steen, High-performance computing systems: Status and outlook, Acta Numerica 21 (2012) 379–474. arXiv:http://journals.cambridge.org/article_S0962492912000050, doi:10.1017/S0962492912000050.
URL http://dx.doi.org/10.1017/S0962492912000050

[13] R. Bellman, R. Bellman, Adaptive Control Processes: A Guided Tour, 'Rand Corporation. Research studies, Princeton University Press, 1961.
URL http://books.google.ch/books?id=POAmAAAAMAAJ

[14] S. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Soviet Math. Dokl. 4 (1963) 240–243.

[15] M. Hegland, Adaptive sparse grids, Anziam Journal 44 (2003) C335–C353.

[16] O. Bokanowski, J. Garcke, M. Griebel, I. Klompmaker, An adaptive sparse grid semi-Lagrangian scheme for first order Hamilton-Jacobi Bellman equations, Journal of Scientific Computing 55 (3) (2013) 575–605, also available as INS Preprint No. 1207. doi:10.1007/s10915-012-9648-x.
URL http://dx.doi.org/10.1007/s10915-012-9648-x

[17] J. Garcke, M. Griebel, Sparse Grids and Applications, Lecture Notes in Computational Science and Engineering Series, Springer-Verlag GmbH, 2012.
URL http://books.google.ch/books?id=__xHU39YLq0C

[18] C. Hager, S. Heber, B. Wohlmuth, Numerical techniques for the valuation of basket options and its greeks, J. Comput. Fin. 13 (4) (2010) 1–31.

[19] H. Rabitz, . Ali, General foundations of highdimensional model representations, Journal of Mathematical Chemistry 25 (2-3) (1999) 197–233. doi:10.1023/A:1019188517934.
URL http://dx.doi.org/10.1023/A%3A1019188517934

[20] M. Holtz, Sparse Grid Quadrature in High Dimensions with Applications in Finance and Insurance, Lecture Notes in Computational Science and Engineering, Springer, Dordrecht, 2011.

[21] A. Murarasu, J. Weidendorfer, G. Buse, D. Butnaru, D. Pflüeger, Compact data structure and parallel alogrithms for the sparse grid technique, in: 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2011.

[22] V. Winschel, M. Kraetzig, Solving, estimating, and selecting nonlinear dynamic models without the curse of dimensionality, Econometrica 78 (2) (2010) 803–821.
URL http://EconPapers.repec.org/RePEc:ecm:emetrp:v:78:y:2010:i:2:p:803-821

[23] K. L. Judd, L. Maliar, S. Maliar, R. Valero, Smolyak method for solving dynamic economic models: Lagrange interpolation, anisotropic grid and adaptive domain, Journal of Economic Dynamics and Control 44 (2014) 92–123.

[24] C. Zenger, Sparse grids, in: W. Hackbusch (Ed.), Parallel Algorithms for Partial Differential Equations, Vol. 31 of Notes on Numerical Fluid Mechanics, Vieweg, 1991, pp. 241–251.
URL http://www5.in.tum.de/pub/zenger91sg.pdf

[25] H.-J. Bungartz, S. Dirnstorfer, Multivariate quadrature on adaptive sparse grids, Computing 71 (2003) 89–114. doi:10.1007/s00607-003-0016-4.
URL http://dx.doi.org/10.1007/s00607-003-0016-4

[26] D. Pflüger, Spatially adaptive sparse grids for high-dimensional problems, Ph.D. thesis, TU München, München (Aug. 2010).
URL http://www5.in.tum.de/pub/pflueger10spatially.pdf

[27] J. Brumm, M. Grill, Computing equilibria in dynamic models with occasionally binding constraints, Journal of Economic Dynamics and Control 38 (2014) 142–160.

[28] L. J. Christiano, J. D. Fisher, Algorithms for solving dynamic models with occasionally binding constraints, Journal of Economic Dynamics and Control 24 (8) (2000) 1179–1232.

[29] T. Hintermaier, W. Koeniger, The method of endogenous gridpoints with occasionally binding constraints among endogenous variables, Journal of Economic Dynamics and Control 34 (10) (2010) 2074–2088.
URL http://ideas.repec.org/a/eee/dyncon/v34y2010i10p2074-2088.html

[30] A. Deftu, A. Murarasu, Optimization techniques for dimensionally truncated sparse grids on heterogeneous systems, in: 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing, IEEE, 2013, pp. 351–358.

[31] M. Heene, C. Kowitz, D. Pflüger, Load balancing for massively parallel computations with the sparse grid combination technique., in: PARCO, 2013, pp. 574–583.

[32] A. Heinecke, D. Pflüger, Emerging architectures enable to boost massively parallel data mining using adaptive sparse grids, International Journal of Parallel Programming 41 (3) (2013) 357–399. doi:10.1007/s10766-012-0202-0.
URL http://dx.doi.org/10.1007/s10766-012-0202-0

[33] P. Hupp, R. Jacob, M. Heene, D. Pflüger, M. Hegland, Global communication schemes for the sparse grid combination technique., in: PARCO, 2013, pp. 564–573.

[34] A. Murarașu, G. Buse, D. Pflüger, J. Weidendorfer, A. Bode, Fastsg: A fast routines library for sparse grids, Procedia Computer Science 9 (2012) 354–363.

[35] A. F. Murarasu, Advanced optimization techniques for sparse grids on modern heterogeneous systems, Ph.D. thesis, Technische Universität München (2013).
URL http://mediatum.ub.tum.de/doc/1137973/1137973.pdf

[36] A. F. Murarasu, J. Weidendorfe, Building input adaptive parallel applications: A case study of sparse grid interpolation, in: Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on, 2012, pp. 1–8. doi:10.1109/ICCSE.2012.11.

[37] D. Pflüger, H.-J. Bungartz, B. Peherstorfer, Density Estimation with Adaptive Sparse Grids for Large Data Sets, SIAM, 2014, Ch. 50, pp. 443–451. arXiv:http://epubs.siam.org/doi/pdf/10.1137/1.9781611973440.51, doi:10.1137/1.9781611973440.51.
URL http://epubs.siam.org/doi/abs/10.1137/1.9781611973440.51

[38] W. Gropp, T. Höfler, R. Thakur, E. Lusk, Using Advanced MPI: Modern Features of the Message-Passing Interface (Scientific and Engineering Computation), MIT Press, 2014.

[39] A. Gaikwad, I. M. Toke, Gpu based sparse grid technique for solving multidimensional options pricing pdes, in: Proceedings of the 2Nd Workshop on High Performance Computational Finance, WHPCF '09, ACM, New York, NY, USA, 2009, pp. 6:1–6:9. doi:10.1145/1645413.1645419.
URL http://doi.acm.org/10.1145/1645413.1645419

[40] E. M. Aldrich, Chapter 10 - {GPU} computing in economics, in: K. Schmedders, K. L. Judd (Eds.), Handbook of Computational Economics Vol. 3, Vol. 3 of Handbook of Computational Economics, Elsevier, 2014, pp. 557 – 598. doi:http://dx.doi.org/10.1016/B978-0-444-52980-0.00010-4.
URL http://www.sciencedirect.com/science/article/pii/B9780444529800000104

[41] Y. Cai, K. L. Judd, G. Thain, S. J. Wright, Solving dynamic programming problems on a computational grid, Computational Economics (2013) 1–24.

[42] Y. Cai, K. L. Judd, T. S. Lontzek, The Social Cost of Carbon with Economic and Climate Risks, ArXiv e-printsarXiv:1504.06909.

[43] N. L. Stokey, R. E. Lucas, Jr., E. C. Prescott, Recursive Methods in Economic Dynamics, Harvard University Press, Cambridge, MA, 1989.

[44] D. K. Backus, P. J. Kehoe, F. E. Kydland, International real business cycles, Journal of political Economy (1992) 745–775.

[45] R. Kollmann, S. Maliar, B. A. Malin, P. Pichler, Comparison of solutions to the multi-country real business cycle model, Journal of Economic Dynamics and Control 35 (2) (2011) 186 – 202, computational Suite of Models with Heterogeneous Agents II: Multi-Country Real Business Cycle Models. doi:http://dx.doi.org/10.1016/j.jedc.2010.09.013.

[46] W. J. Den Haan, K. L. Judd, M. Juillard, Computational suite of models with heterogeneous agents ii: Multi-country real business cycle models, Journal of Economic Dynamics and Control 35 (2) (2011) 175–177.

URL http://ideas.repec.org/a/eee/dyncon/v35y2011i2p175-177.html

[47] F. E. Kydland, E. C. Prescott, Time to build and aggregate fluctuations, Econometrica: Journal of the Econometric Society (1982) 1345–1370.

[48] A. Klimke, B. Wohlmuth, Algorithm 847: Spinterp: piecewise multilinear hierarchical sparse grid interpolation in matlab, ACM Trans. Math. Softw. 31 (4) (2005) 561–579. doi:10.1145/1114268.1114275.
URL http://doi.acm.org/10.1145/1114268.1114275

[49] F. Barillas, J. Fernndez-Villaverde, A generalization of the endogenous grid method, Journal of Economic Dynamics and Control 31 (8) (2007) 2698 – 2712. doi:http://dx.doi.org/10.1016/j.jedc.2006.08.005.
URL http://www.sciencedirect.com/science/article/pii/S0165188906001783

[50] W. L. Maldonado, B. Svaiter, Hölder continuity of the policy function approximation in the value function approximation, Journal of Mathematical Economics 43 (5) (2007) 629–639.

[51] O. F. Morand, K. L. Reffett, Existence and uniqueness of equilibrium in nonoptimal unbounded infinite horizon economies, Journal of Monetary Economics 50 (6) (2003) 1351–1373.

[52] M. Datta, L. J. Mirman, O. F. Morand, K. L. Reffett, Markovian equilibrium in infinite horizon economies with incomplete markets and public policy, Journal of Mathematical Economics 41 (4) (2005) 505–544.

[53] A. Waechter, L. T. Biegler, On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming, Math. Program. 106 (1) (2006) 25–57. doi:10.1007/s10107-004-0559-y.
URL http://dx.doi.org/10.1007/s10107-004-0559-y

[54] L. Landau, E. Lifšic, P. Ziesche, L. Pitaevskij, Hydrodynamik, Lehrbuch der theoretischen Physik : in 10 Bänden / L.D. Landau; E.M. Lifschitz. In dt. Sprache

hrsg. von Paul Ziesche, Deutsch, 2007.

URL http://books.google.com/books?id=fbYwvvqoKScC