Optimizing High-Dimensional Dynamic Stochastic Economic Models for MPI+GPU Clusters

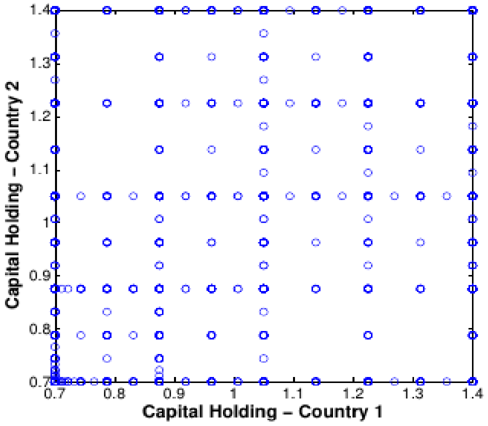
Simon Scheidegger

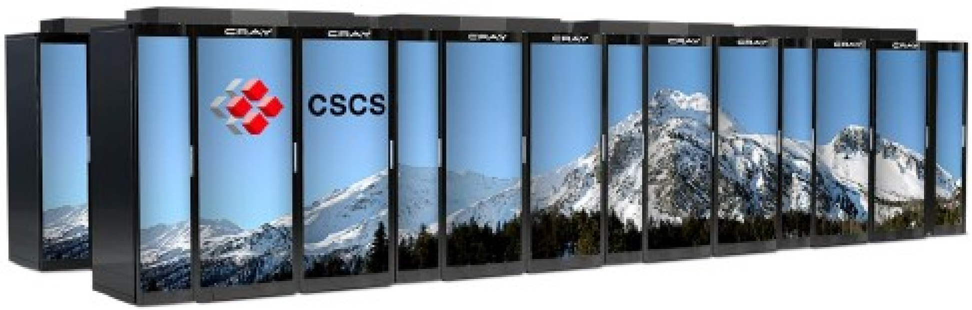
GTC – San Jose, March 17-20th, 2015

Johannes Brumm (UZH)

Dmitry Mikushin (USI LUGANO & https://parallel-computing.pro/)

Olaf Schenk (USI LUGANO)





Outline

0.) Dynamic Stochastic Economic Models

I.) From Full (Cartesian) Grids to Sparse Grids

II.) Adaptive Sparse Grids

III.) Time Iteration, Adaptive Sparse Grids & HPC

(Macro-) Economic Models

e.g. Stokey, Lucas & Prescott (1989), Ljundquist & Sargent (2004), Krüger & Kübler (2004), Judd et. al. (2013),...

- International Real Business Cycle (IRBC) Models:
Exchange Rates, Global Trade Imbalances

- Dynamic Stochastic General Equilibrium (DSGE) Models:
Monetary Policy, Business Cycle Fluctuations

- Overlapping Generations (OLG) Models:
Demographic Change, Social Security

→ *Disclaimer: when I talk about Economics, I am not concerned with financial mathematics, financial engineering (option pricing, estimation of financial data,...)*

e.g. Hager (2010), Holtz (2011), Heinecke et. al (2013), Winschel & Krätzig (2010),...

Our motivation

i) Economic models: heterogeneous & high-dimensional (e.g. IRBC)

ii) Want to solve dynamic stochastic models with high-dimensional state spaces:

$$\text{“}\Theta v = v\text{”} \longleftrightarrow |\Theta v_i - v_{i+1}| < \varepsilon$$

→ Have to interpolate high-dimensional functions

Problem: curse of dimensionality
→ N^d points in ordinary discretization schemes

iii) **Want to overcome curse of dimensionality**

iv) **Want locality & adaptivity of interpolation scheme**

iv) **Speed-up** → access hybrid HPC systems (MPI, OpenMP, GPU)

Example: Infinite-Horizon Dynamic Programming

e.g. Stokey, Lucas & Prescott (1989), Judd (1998), ...

Want to choose an infinite sequence of "controls" $\{u_s\}_{s=0}^{\infty}$ to maximize

$$\sum_{t=0}^{\infty} \beta^t r(x_t, u_t) \quad \text{s.t.} \quad x_{t+1} = g(x_t, u_t) \qquad \beta \in (0,1)$$

(Discrete time) Dynamic programming seeks a **time-invariant policy function *h*** mapping the state x_t into the control u_t, such that the sequence $\{u_s\}_{s=0}^{\infty}$ generated by iterating

$$u_t = h(x_t)$$

$$x_{t+1} = g(x_t, u_t)$$

starting from an initial condition solves the original problem.

r in the economic context: often a so-called `utility function'.
r concave: reflects the notion "more is better"; marginal benefit tends to zero.

Value Function Iteration

The solution is approached in the limit as $j \rightarrow \infty$ by iterations on at every coordinate of the discretized grid.

$$V_{j+1}(x) = \max_u \{ r(x,u) + \beta V_j(\tilde{x}) \}$$

s.t.

$$\tilde{x} = g(x,u)$$

x: grid point, describes your system. State-space potentially **high-dimensional.**

`old solution': high-dimensional function, approximated by sparse grid Interpolation method on which we Interpolate.

Use-case for (adaptive) sparse grids

Dynamic Economic Models as Functional Equations

$x_t \in X \subset \mathbb{R}^d$ **State of the economy at time *t*** (e.g. capital holdings of different countries).

→ State influences agent's dynamic behaviour (e.g. investment choices for each country).

$policy\ function\ p : X \to Y$ *Y*: **space of possible policies** (e.g. investment choices).

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

Transition of the economy from one period to the next
→ depends on the *current state and policies*.
→ distribution *F* is given.

→ ***p*** is a **solution** to the following type of ***functional equation***:

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

E is given by period-to-period *Equilibrium conditions* of the model.
→ solve for **p** by backward iteration → time iteration (**'fixpoint problem'**):

Interpolation on a Full Grid

-Consider a **d-dimensional function** $f : \Omega \to \mathbb{R}$ on Ω = [0,1]^d

-In numerical simulations:
f might be expensive to evaluate! (Optimization/system of non-linear Eqs.)

-But: need to be able to evaluate *f* at arbitrary points using a numerical code (since we iterate on 'old' solution)

-Construct an interpolant ***u*** of ***f*** $f(\vec{x}) \approx u(\vec{x}) := \sum_i \alpha_i \varphi_i(\vec{x})$

-With suitable basis functions: $\varphi_i(\vec{x})$

-and coefficients: α_i

Basis Functions & Hierarchical Increment Spaces

Hierarchical increment spaces:

$$W_l := \text{span}\{\phi_{l,i} : i \in I_l\}$$

with the **index set I_l** (*i* almost always 'odd')

The corresponding function space:

$$V_l = \bigoplus_{k \le l} W_k$$

The **1d-interpolant** (multi-d: Tensor product)

$$f(x) \approx u(x) = \sum_{k=1}^{l} \sum_{i \in I_k} \alpha_{k,i} \phi_{k,i}(x)$$

!!MOVIE!!

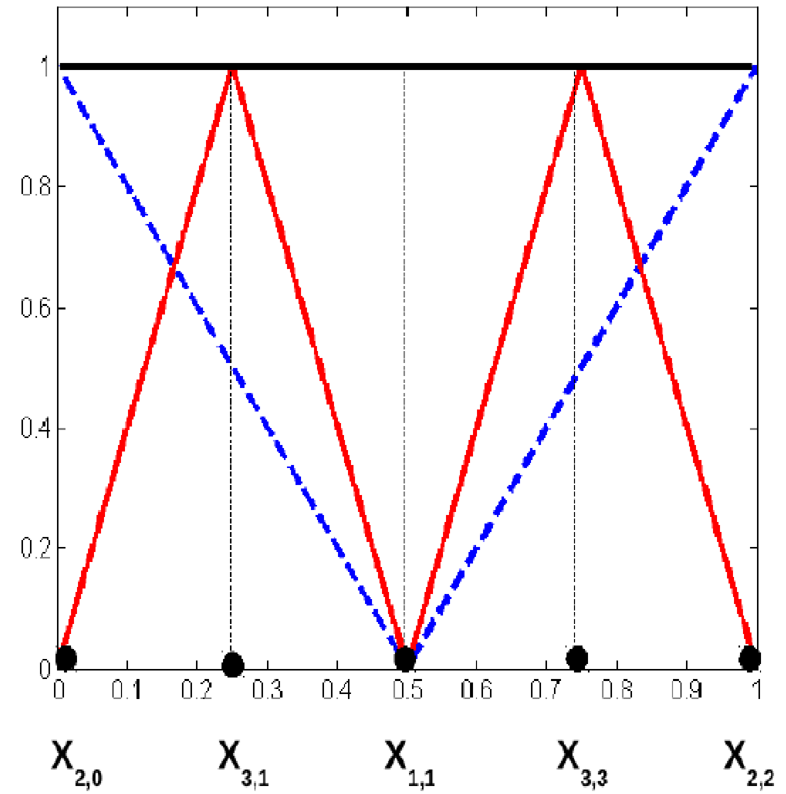


Fig.: 1-d basis functions $\phi_{l,i}$ and the corresponding **grid points** up level ***l* = 3** in the hierarchical basis. support of all basis functions of **W_k mutually disjoint!**

Why reality bites...

Interpolant consists of ***O(2^nd^)*** grid points

For **sufficiently smooth *f*** and its interpolant ***u***, we obtain an asymptotic error decay of $\|f(\vec{x}) - u(\vec{x})\|_{L_2} \in \mathcal{O}\left(h_n^2\right)$

But at the cost of $\mathcal{O}\left(h_n^{-d}\right) = \mathcal{O}\left(2^{nd}\right)$

function evaluations → **"curse of dimensionality"**

Hard to handle more than 4 dimensions numerically

→ e.g. d=10, n = 4, 15 points/d, **5.8×10^{11}** grid points

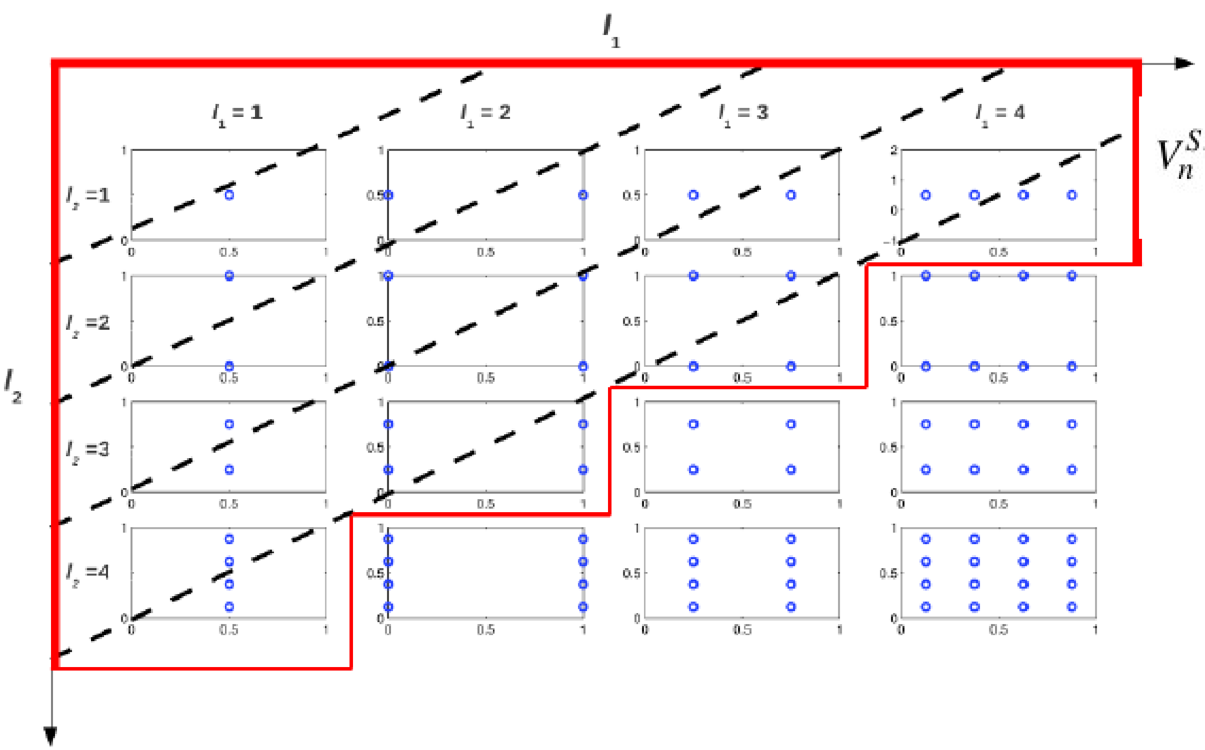
`Breaking' the curse of dimensionality II

(see, e.g. Bungartz & Griebel (2004))

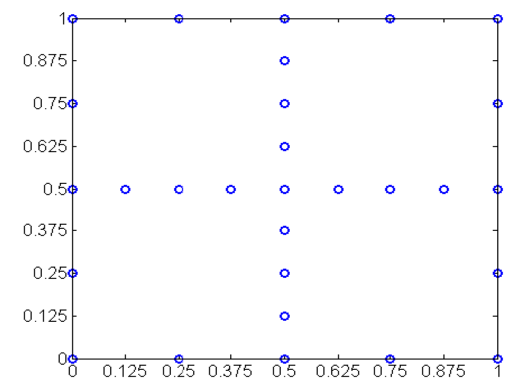
-Strategy for constructing sparse grid: leave out those subspaces from full grid that only contribute little to the overall interpolant.

- if second mixed derivatives are bound: $|\alpha_{\vec{l},\vec{i}}| = \mathcal{O}\left(2^{-2|\vec{l}|_1}\right)$

-Optimization w.r.t. number of degrees of freedom (grid points) and the approximation accuracy leads to the sparse grid space of level ***n***.



$$V_n^{S,CC} := \bigoplus_{|\vec{l}|_1 \le n+d-1} W_{\vec{l}}.$$



Sparse grid: 29 pt.
Full grid : 81 pt.

100d Sparse grid: 1,394,001
100d Full grid > Googol

Adaptive Sparse Grids

See, e.g., Bungartz (2003), Ma & Zabaras (2008), Pflüger (2010),...

-Surpluses quickly decay to zero as the level of interpolation increases assuming a smooth fct.

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

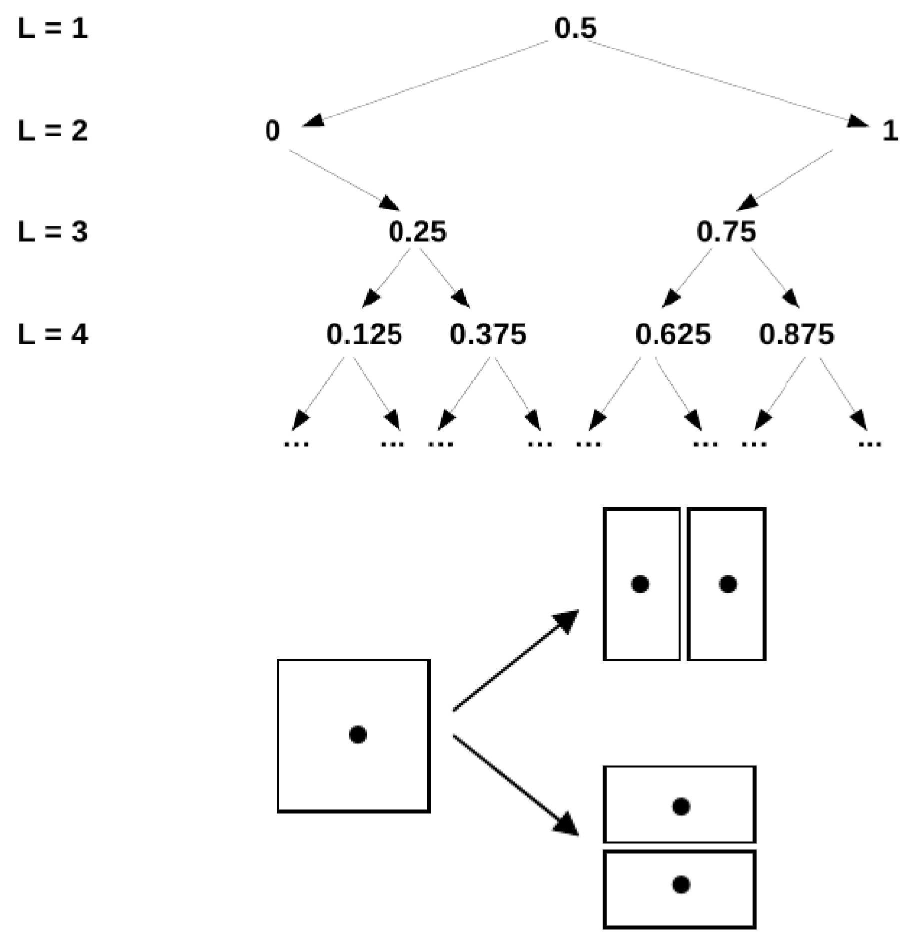
-Use hierarchical surplus as error indicator.

-`Automatically' detect regions of high curvature and adaptively refine the points.

-Each grid point has ***2d*** neighbours

-Add neighbour points, i.e. locally refine interpolation level from *l* to *l+1*

-Criterion: **e.g.** $|\alpha_{\vec{l},\vec{i}}| \geq \epsilon$



top panel: tree-like structure of sparse grid.
lower panel: locally refined sparse grid in 2D.

Test in 2d (Movie)

Test function: $$\frac{1}{|0.5 - x^4 - y^4| + 0.1}$$

Max. Error:
O(10^{-2})

Full grid:
→ *O(10^{9})* points

Sparse grid:
→ **311'297 points**

Adaptive sparse grid:
→ **4'411 points**

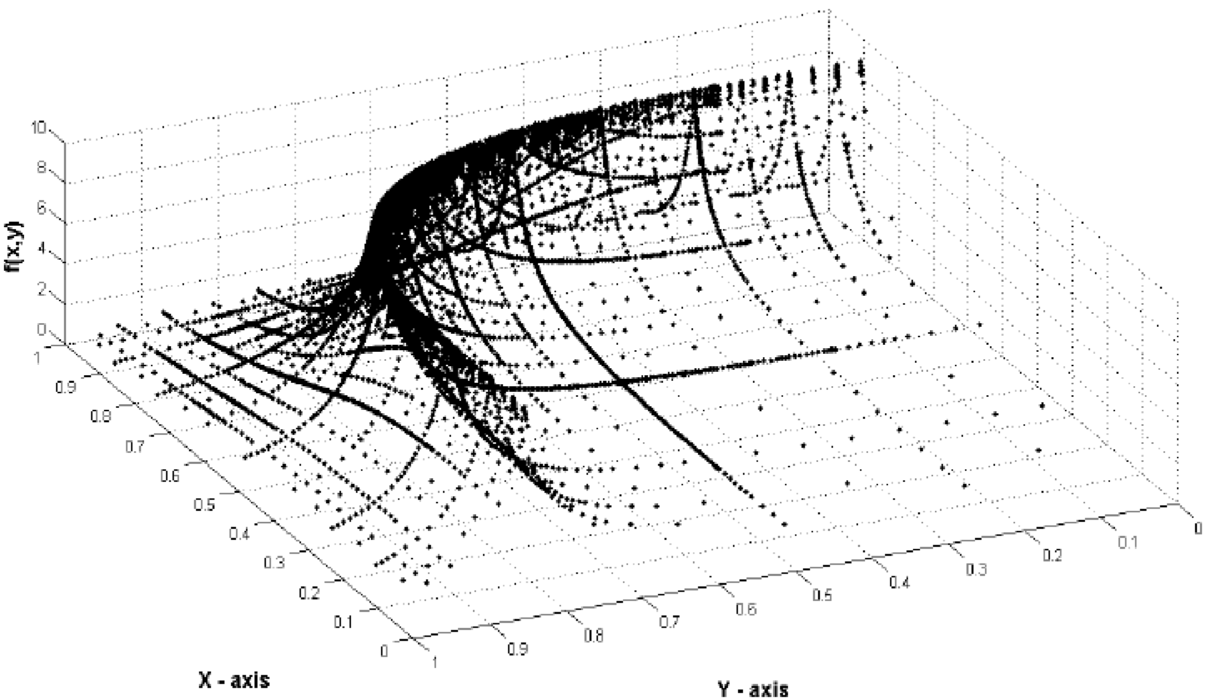


Fig.: 2d test function and its corresponding grid points after 15 refinement steps.

IRBC: Model ingredients

RECALL: WE WANT TO SOLVE HIGH-DIM MODELS

- Use standard problem for testing comp. methods for high-dim problems.
→ **I**nternational **R**eal **B**usiness **C**ycle model (**IRBC**) with adjustment costs

(e.g. Den Haan et al. (2011), Malin et al. (2011))

- *N* countries facing productivity shocks and capital adjustment costs

→ they differ wrt. **productivity** (stochastic and exogen.) '***a***' & **capital stock** (endogen.) '***k***'

→ dimension of the state space / grid: **dim=2N**

→ one Euler equation per country plus aggregate resource constraint:
N+1 equations characterize equilibrium at each point

- Use time iteration to solve for the optimal policy $p: R_+^{2N} \rightarrow R_+^{N+1}$

- Some Models → in each country, **investment is irreversible.**
installed capital cannot be consumed or moved to another country $p: R_+^{2N} \rightarrow R_+^{2N+1}$

Parallel time iteration/DP algorithm

Brumm, Scheidegger (2014 – revise & resubmit); Brumm, Mikushin, Scheidegger, Schenk (2014 – revise & resubmit)

-Our implementation:

Hybrid parallel

(MPI & Intel TBB & GPU (CUDA/THRUST)).

-newly generated points are distributed via MPI

Solve optimizations/ nonlinear equations locally

(e.g. IPOPT (Waechter & Biegler (2006)).

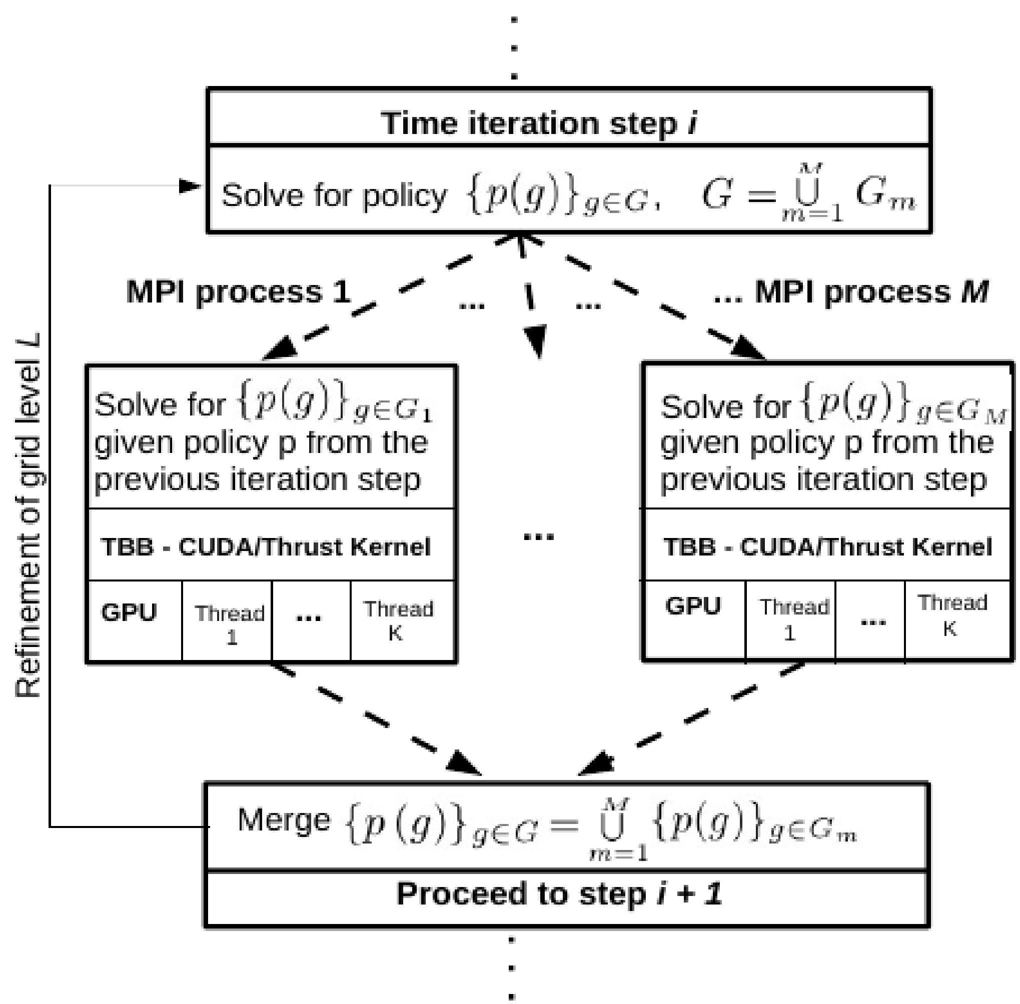
In parallel: `messy' !

→ policy from previous iteration visible on all MPI processes.

→ we have to ensure some sort of `load balancing'.

→ **Now a lot better with TBB**

One single time-step



GPU Optimizations

Brumm, Mikushin, Scheidegger, Schenk (2014 – revise & resubmit)

I) Simplify arithmetic expressions, eliminate divisions (most expensive)

ii) Eliminate duplicate computations, keeping the same byte per FLOP ratio, eliminate branching

iii) Parallelize function evaluation with Thrust using combined transform+reduce (transform_reduce)

iv) Eliminate redundant cudaMalloc/cudaFree from Thrust implementation

v) Runtime optimization: hard-code vector size into **GPU** kernel and pass vector elements as scalars, together with other kernel arguments
→ 15% perf improvement, but needs JIT-compilation
→ Stores compiled kernels onto disk ⇒ could be slower on cluster, requires singleton for MPI/threads

vi) Hybrid multi-threading with Intel TBB: (N − 1) threads on **CPU**, 1 thread - for **GPU**; TBB balances workloads automatically with “work stealing”

vii) Vectorize **CPU** kernel with AVX

Intel® Threading Building Blocks (TBB)

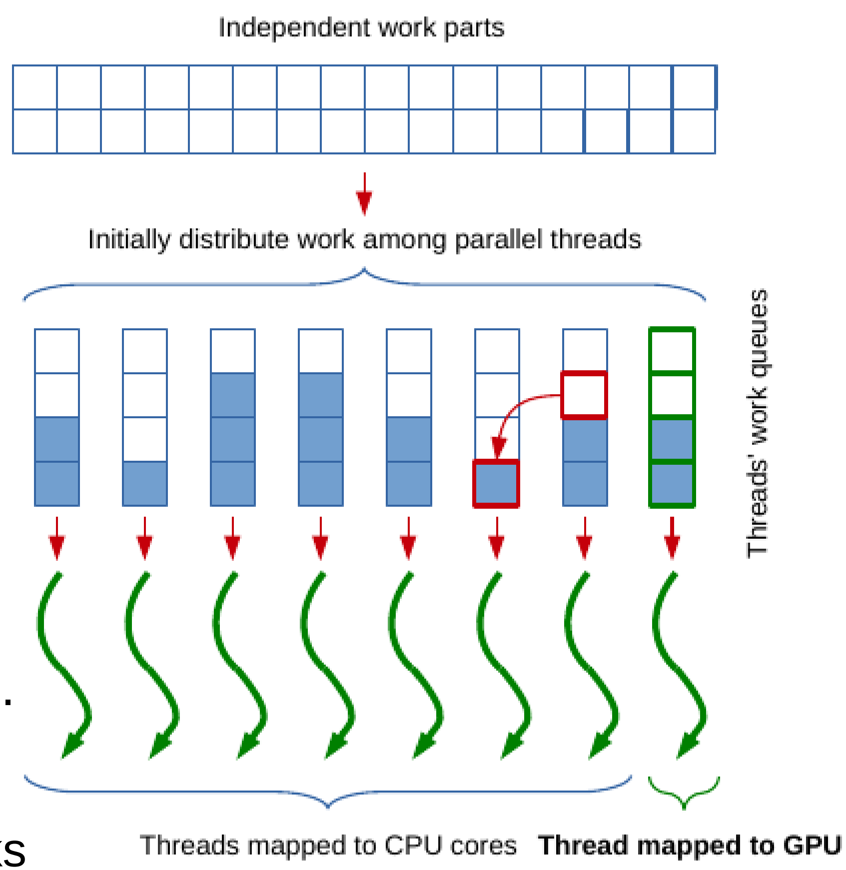
-TBB maps different threads, similar to OpenMP.

-Every thread is initially assigned an equal logical queue of tasks.

-However, different tasks may be processed faster or slower, due to differences between tasks and/or compute cores

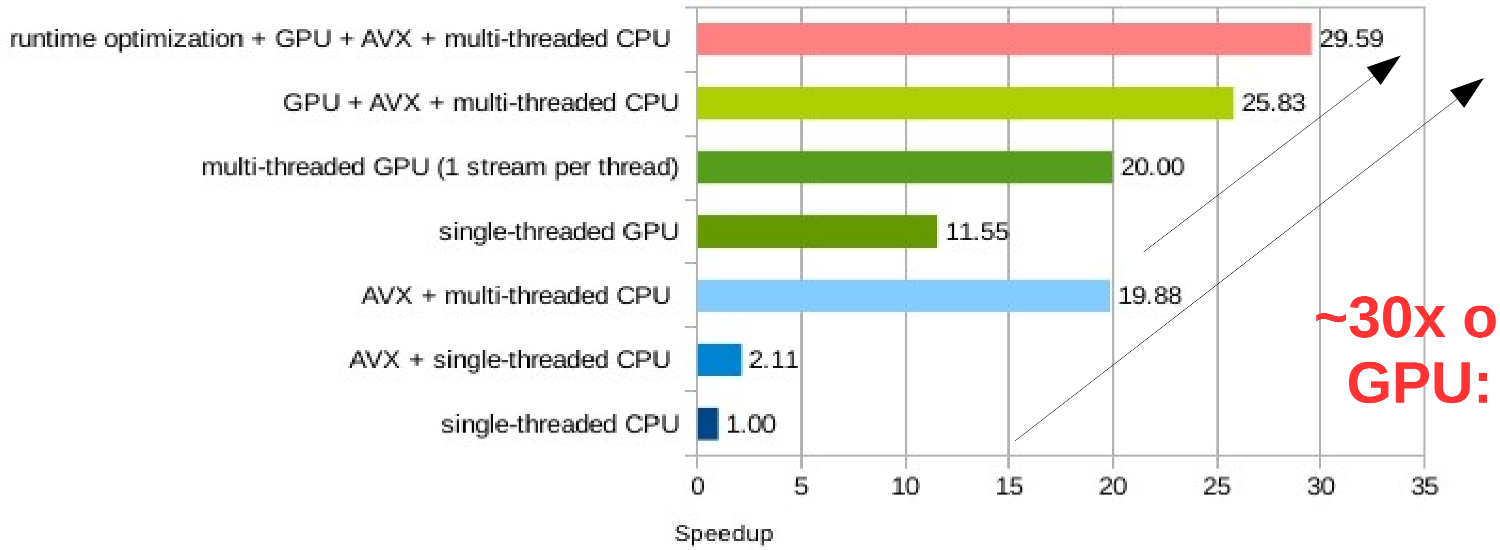
-<u>TBB approach to work balancing</u>: once one thread runs out of tasks, **“steal” a task from another thread, which makes slower progress**.

“hddm-solver” maps one extra thread onto **GPU** → **CPU** cores and **GPU** process interpolation tasks together.

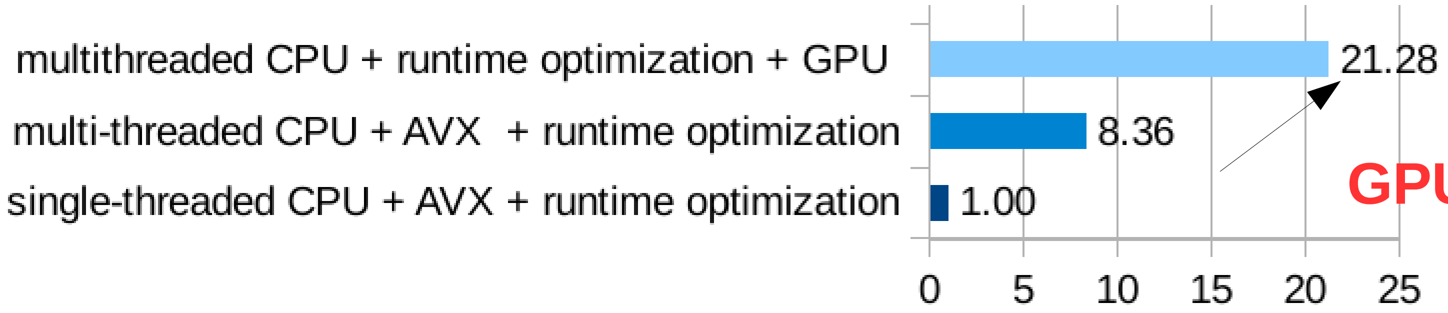


Single-node Code Optimization

Brumm, Mikushin, Scheidegger, Schenk (2014 – revise & resubmit)



~30x on node
GPU: 50%



GPU: 250%

A Strong Scaling Example

nodes are equipped with an 8-core 64-bit Intel SandyBridge CPU (Intel® Xeon® E5-2670), an NVIDIA® Tesla® K20X

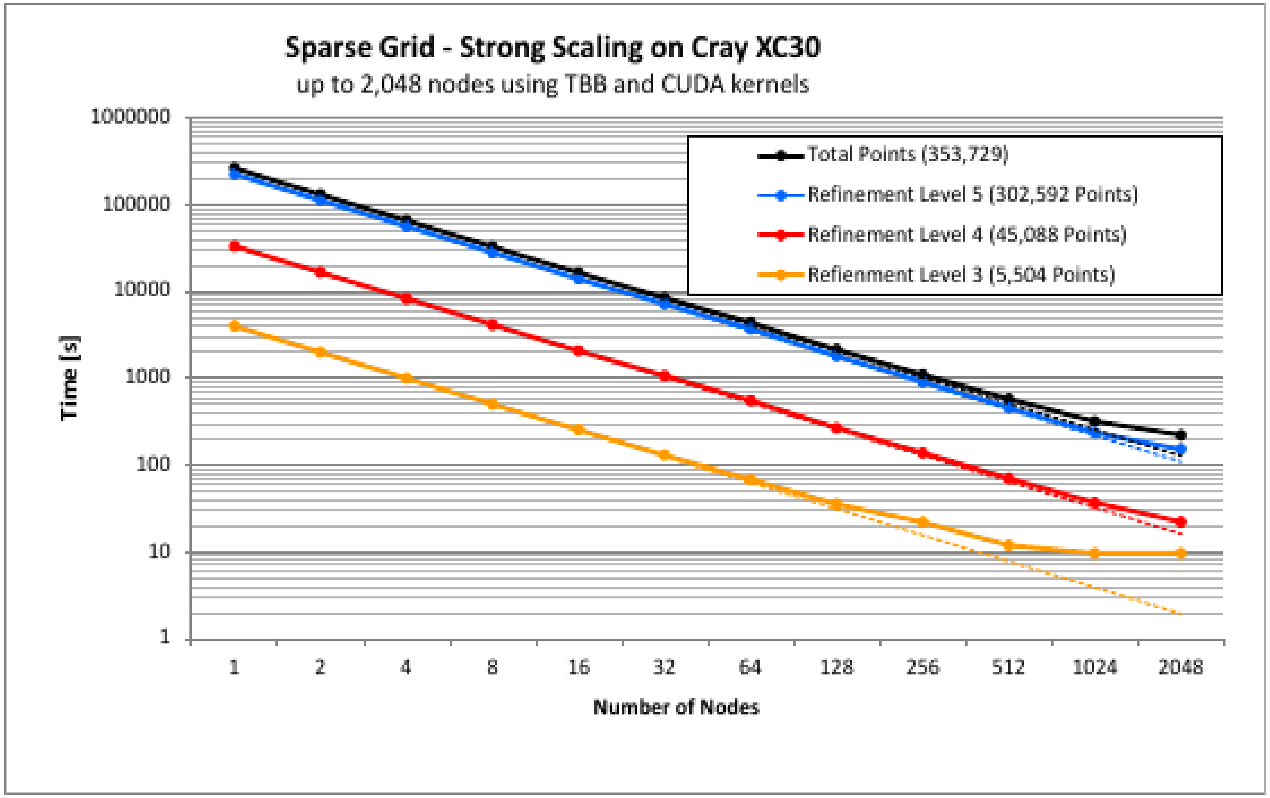


Fig.: strong scaling of the code.
Problem: one timestep of an 16d IRBC, fixed refinement level 5.

Econ. Example: Results (log10)

Euler Errors: At optimal solution, marginal benefit of consuming one unit of output today is equal to the marginal benefit of investing it. Euler errors measures how much they deviate.

Increase Dimension
→ errors remain
of same quality

Dimension	Level	Points	Max. Error	Avg. Error
4	3	41	-2.95	-3.18
8	3	145	-3.04	-3.25
12	3	313	-2.81	-3.27
16	3	545	-2.59	-3.29
20	3	841	-2.93	-3.30
50	3	5,101	-2.64	-3.33
100	3	20,201	-2.79	-3.33

Asset Pricing Example

- Want to study economic implications of more countries in the model.

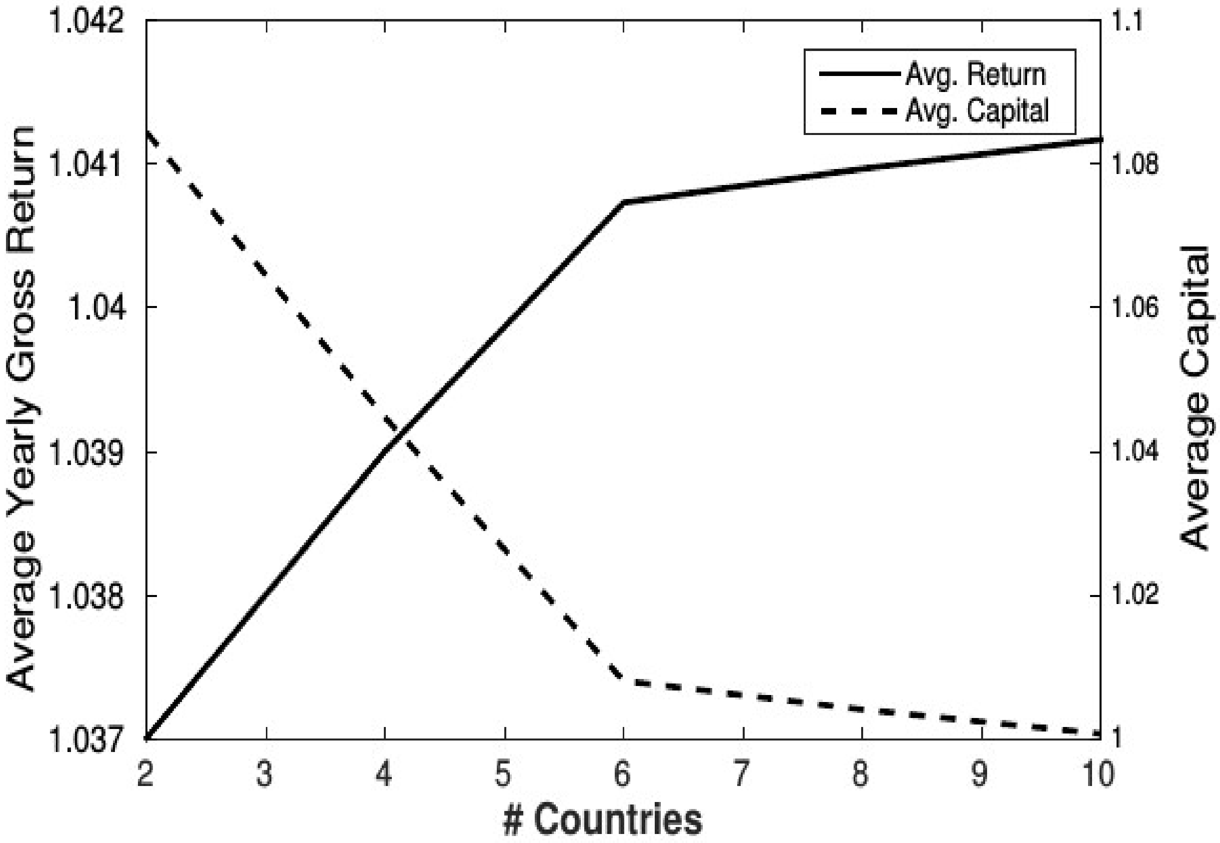
- We keep the model symmetric:
  →All countries: same exposure to risk (country & global shocks)

$$\gamma^j = 0.25$$

- diversification effect:
  → **risks** faced by each country become **less severe** as the number of countries increases.
  →the **countries save less** (lower precautionary savings)
  → lower capital stocks

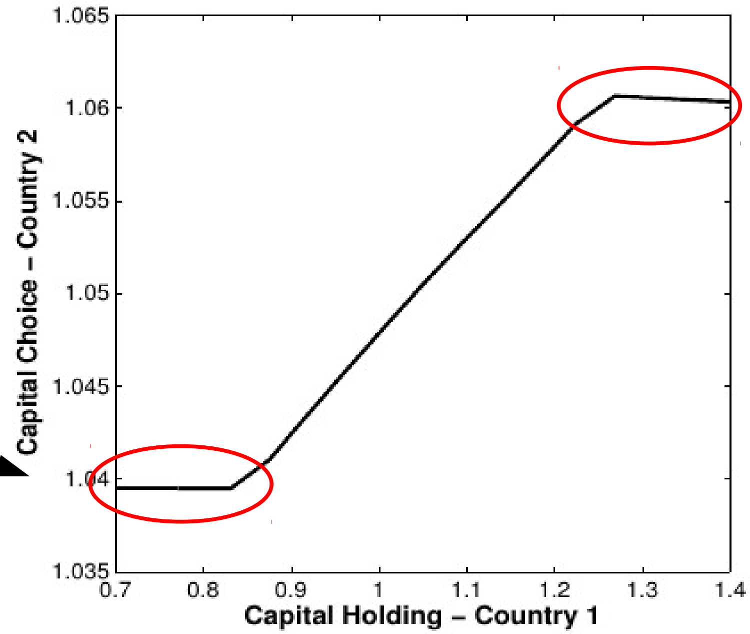
- higher mean returns:
  → Because of decreasing marginal returns to capital, lower capital implies higher returns.

  → More countries: better chances to reallocate capital where it is more productive and yields higher returns.



IRBC with irreversible investment

Capital in country 1 low:
→ investment opportunities better than in country 2
→ irrev. constraint binding for country 2.



Capital in country 1 high:
→ irrev. constraint binding for country 1.
→ this limits transfer of resources to country 2.

Fig.: Capital choice of country 2 as a function of capital holding of country 1.
All other state variables of this model are kept fixed at steady state (2N = 4d).
The 4-d policy function was interpolated on an adaptive sparse grid (ε = 0.0033).

Note: **kink is (2N -1) - dimensional hypersurface in 2N - dim state space.**

Models with binding constrains: massive speedup due to adaptivity

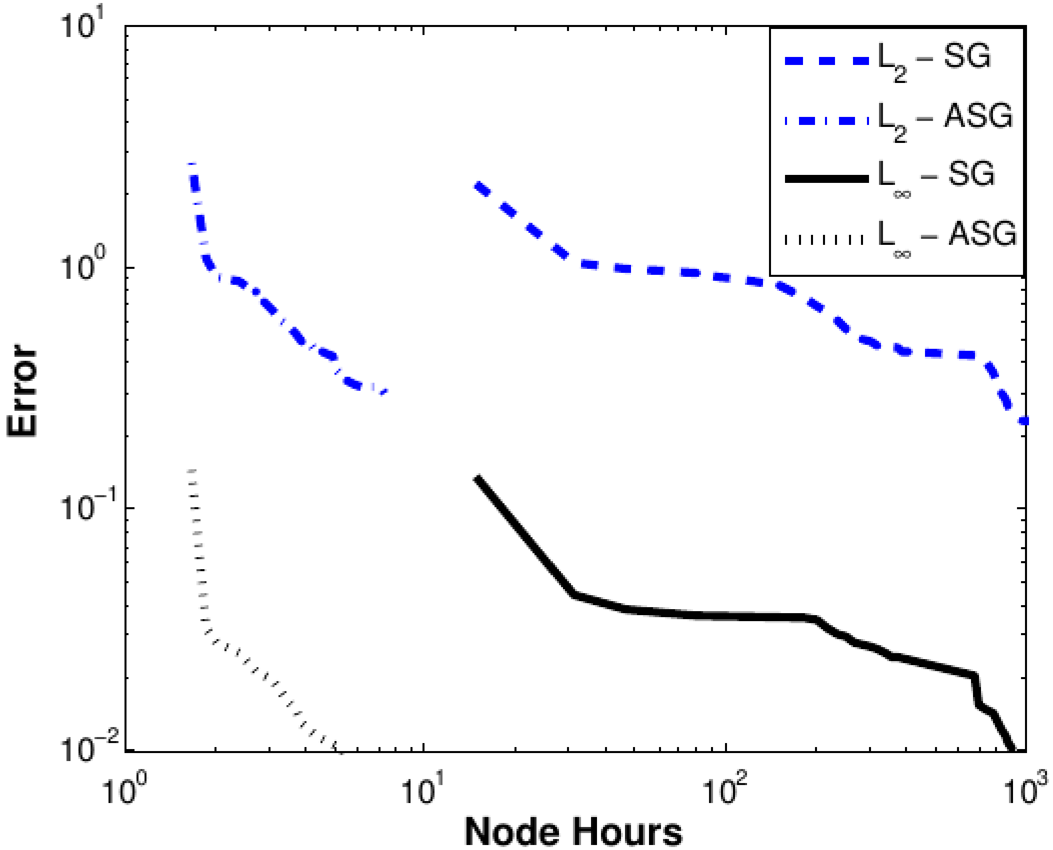


Fig.: 8d model with binding constraints.
model run with/without adaptive sparse grids.
Relative error among two consecutive time-steps.
10k points drawn from uniform distribution.

Dimension	Points	Max. Error	Avg. Error	$\lvert V_8^{S,CC} \rvert$
4	245	-2.22	-2.88	18,945
6	684	-2.26	-2.73	127,105
8	931	-2.02	-2.66	609,025
10	2,790	-1.97	-2.54	2,148,960
12	4,239	-1.81	-2.48	7,451,394
16	8,569	-1.94	-2.36	52,789,761
20	9,098	-1.96	-2.35	$\gg 10^8$

Tab.: Comparison of a sparse and adapt. sparse grid of comparable accuracy.

Summary & Conclusion

- **Algorithm perfectly suited to solve high dimensional dynamic models with large amount of heterogeneity! (Method: Scalable & Flexible).**
- First time adaptive sparse grids are applied to high-dimensional economic models.
- First ones to solve dynamic economic models on HPC systems with hybrid (MPI, TBB, GPU) parallelism.
- GPU can speed-up application up to 2-3x

Can now address:

- International Real Business Cycle (IRBC) Models: **Exchange Rates, Global Trade Imbalances**
- Dynamic Stochastic General Equilibrium (DSGE) Models: **Monetary Policy, Business Cycle Fluctuations**
- Overlapping Generations (OLG) Models: **Demographic Change, Social Security**

Contact Details

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