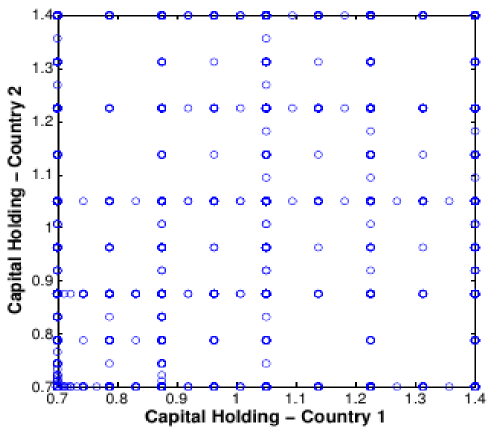
Sparse Grids

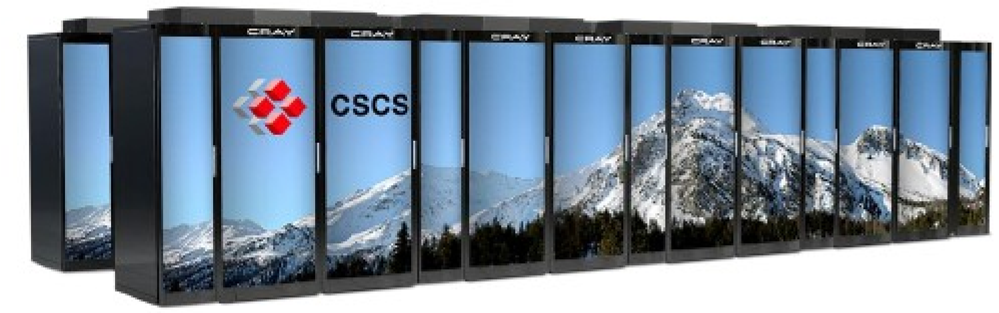
Simon Scheidegger

ZICE, University of Zürich, Jan 24th – Feb. 1st, 2017

Brumm & Scheidegger (2014)

Preprint: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2349281





Outline

I. Motivation – “the curse of dimensionality”

II. From Full (Cartesian) Grids to Sparse Grids

III. Adaptive Sparse Grids

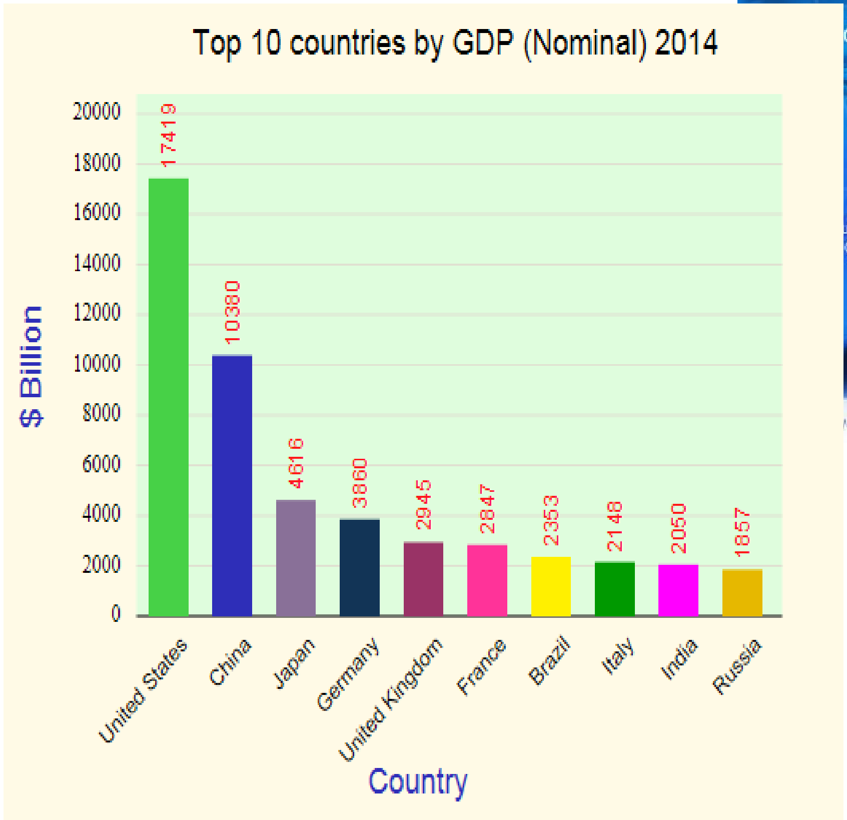
IV. Putting things together:
Time Iteration, Adaptive Sparse Grids &
High-Performance Computing (HPC)

Example – Heterogeneity in IRBC models

- Model trade imbalance

- FX rates

-...



- How many regions does a minimal model have?
- Are policy functions smooth? (borrowing constraints)

→ **Model heterogeneous & high-dimensional**

Example – Heterogeneity in OLG* models

*Overlapping generation models

To model e.g. social security:

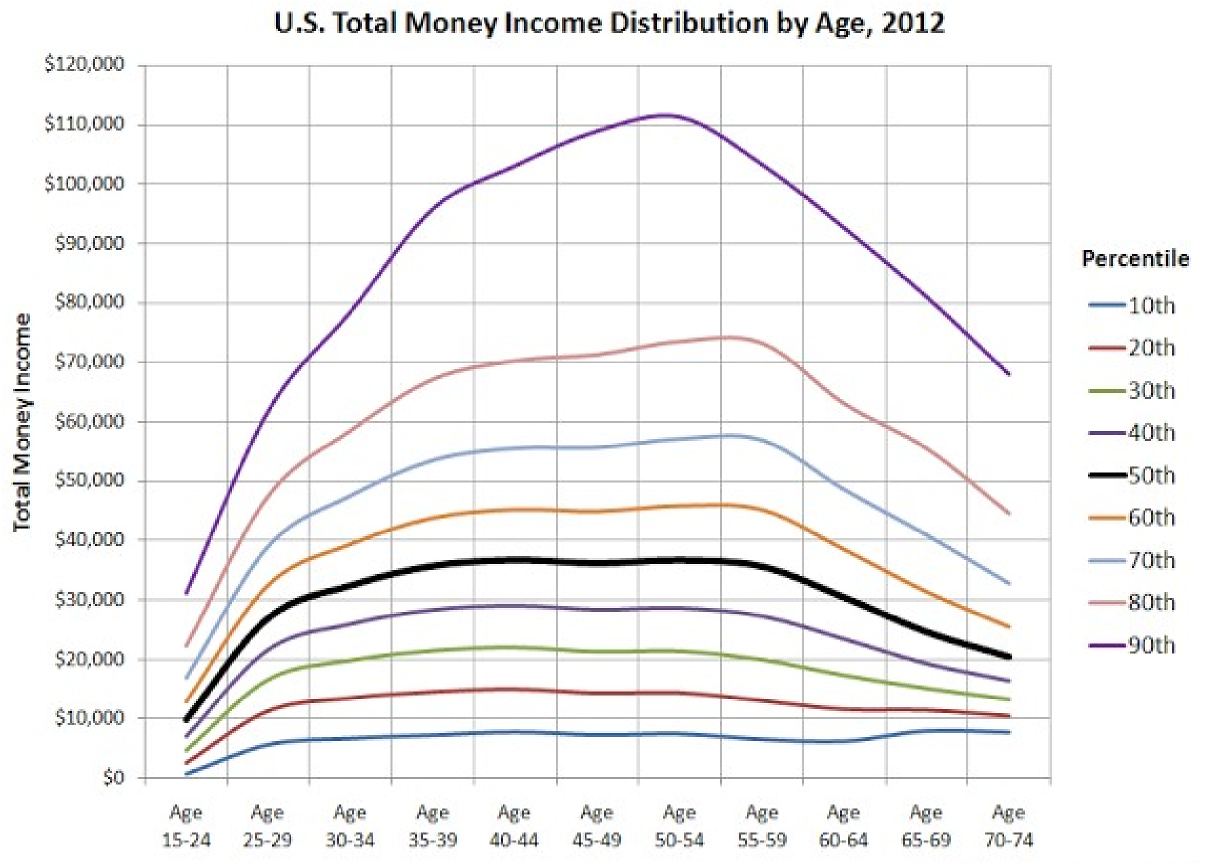
- How many age groups?

- borrowing constraints?

- aggregate shocks?

- …

→ Model: heterogeneous & high-dimensional



Financial markets: non-Gaussian returns

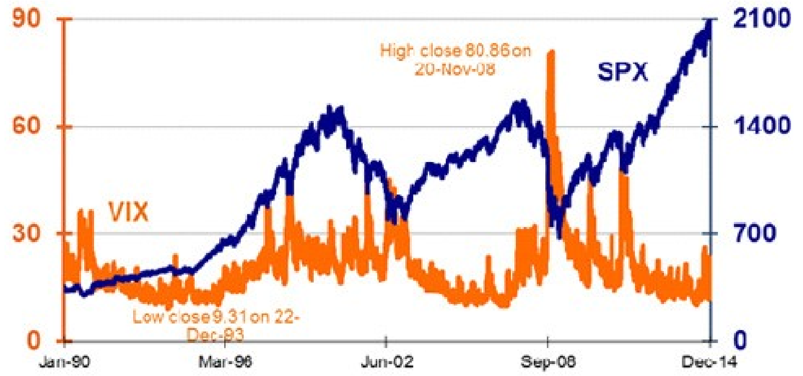
- Derivative contracts giving a right to buy or sell an underlying security.
    - *European* if exercise at expiration only.
    - ***American*** if exercise any time until expiration.

- American options are extremely challenging:
  → Dynamic optimization problem.

- Basic models do not describe dynamics accurately (e.g., Hull (2011)).

- Financial returns are often not Gaussian.

- Realistic models are hard to deal with, as they need many factors.
  → Curse of dimensionality.



Dynamic Programming/Value Function Iteration

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

Dynamic programming seeks a time-invariant policy function ***p*** mapping a state $\mathbf{x}_t$ into the control $\mathbf{u}_t$ such that for all $t \in \mathbb{N}$ $u_t = p(x_t)$
The solution is approached in the limit as $j \longrightarrow \infty$ by iterations on:

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

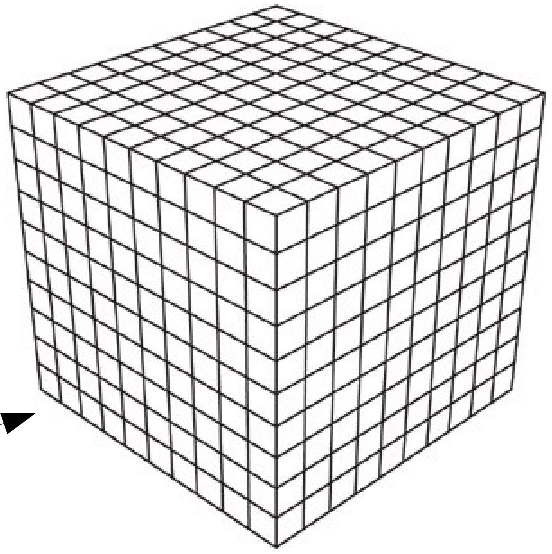
s.t.

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

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

`old solution':
high-dimensional function on which **we interpolate**.

→**N^d** points in ordinary discretization schemes.

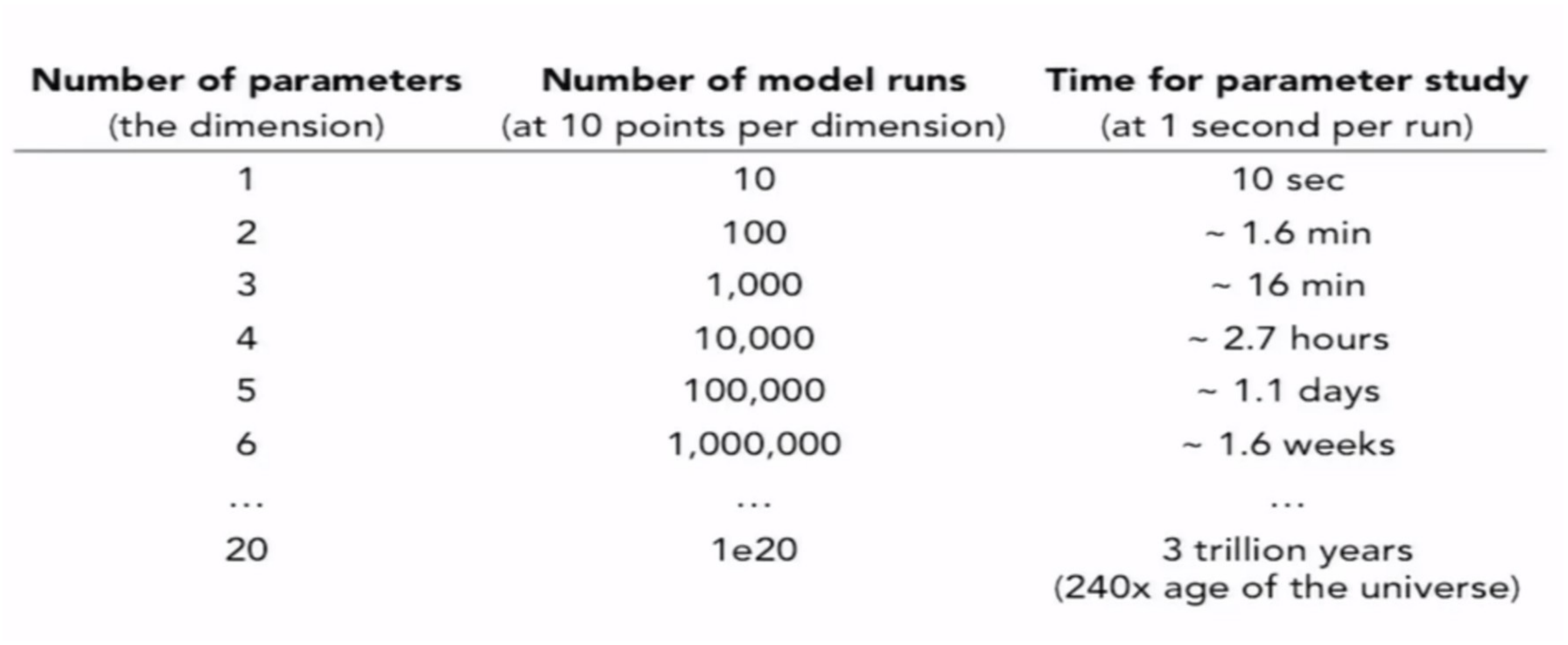


→Use-case for (adaptive) sparse grids.

How many is dimensions is high dimensions?

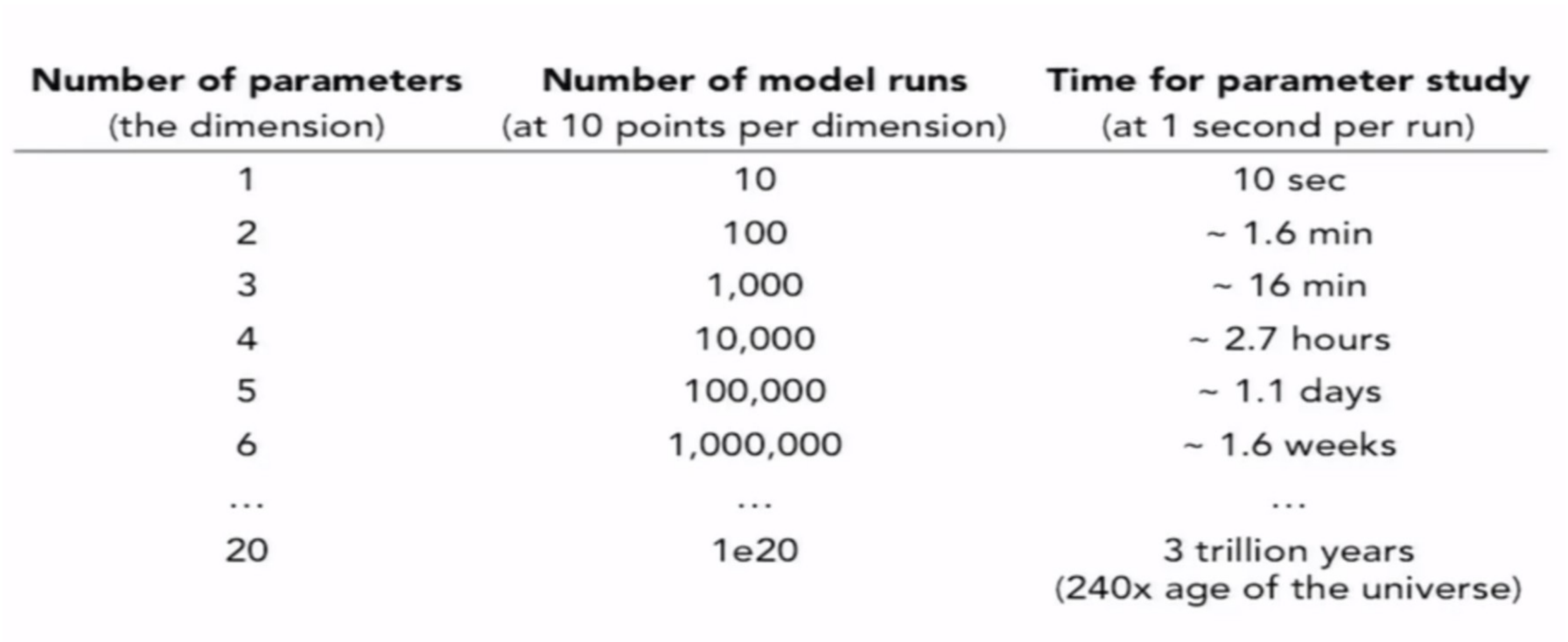
?

How many is dimensions is high dimensions?



Number of parameters (the dimension)	Number of model runs (at 10 points per dimension)	Time for parameter study (at 1 second per run)
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
...	...	...
20	1e20	3 trillion years (240x age of the universe)

How many is dimensions is high dimensions?



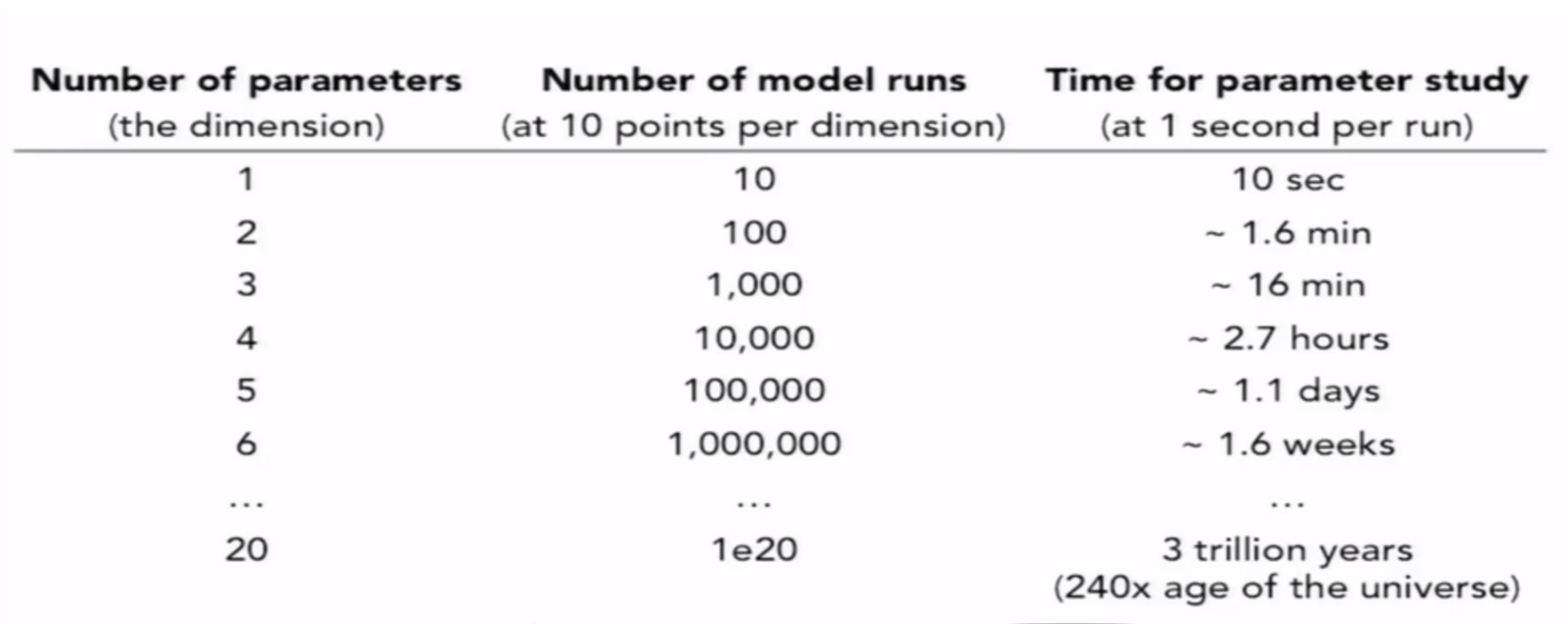
Number of parameters (the dimension)	Number of model runs (at 10 points per dimension)	Time for parameter study (at 1 second per run)
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
...	...	...
20	1e20	3 trillion years (240x age of the universe)

Dimension reduction
Exploit symmetries,...

Deal with #Points
Adaptive Sparse Grids

High-performance computing
Reduces time to solution, but not the problem size

How many is dimensions is high dimensions?



Number of parameters (the dimension)	Number of model runs (at 10 points per dimension)	Time for parameter study (at 1 second per run)
1	10	10 sec
2	100	~ 1.6 min
3	1,000	~ 16 min
4	10,000	~ 2.7 hours
5	100,000	~ 1.1 days
6	1,000,000	~ 1.6 weeks
...	...	...
20	1e20	3 trillion years (240x age of the universe)

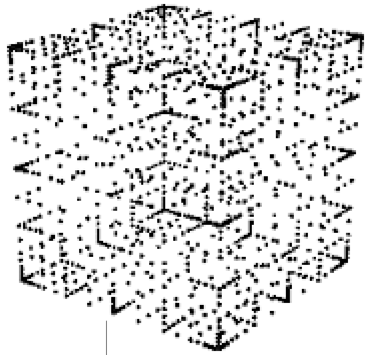
Dimension reduction
Exploit symmetries,...

Deal with #Points
Adaptive Sparse Grids

High-performance computing
Reduces time to solution, but not the problem size

Computational modelling

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



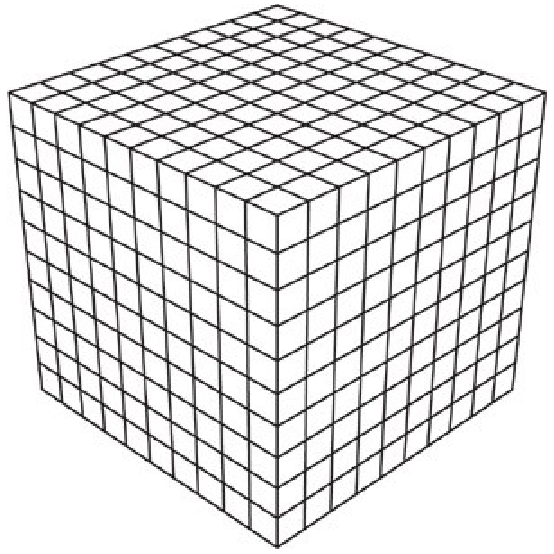
Abstract Problem Formulation

i) Dynamic models: heterogeneous & high-dimensional

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

→ Have to approximate and 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**

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

Models where high-dim. state spaces show up

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

•Mathematical Finance:
Option pricing,...

Our Method

Adaptive Sparse Grids:

- Alleviate curse of dimensionality by sparsity.
- Capture non-smoothness by adaptivity.

Dynamic Programming (DP) / Time iteration:

- Iteratively update e.g. value functions or policy functions.

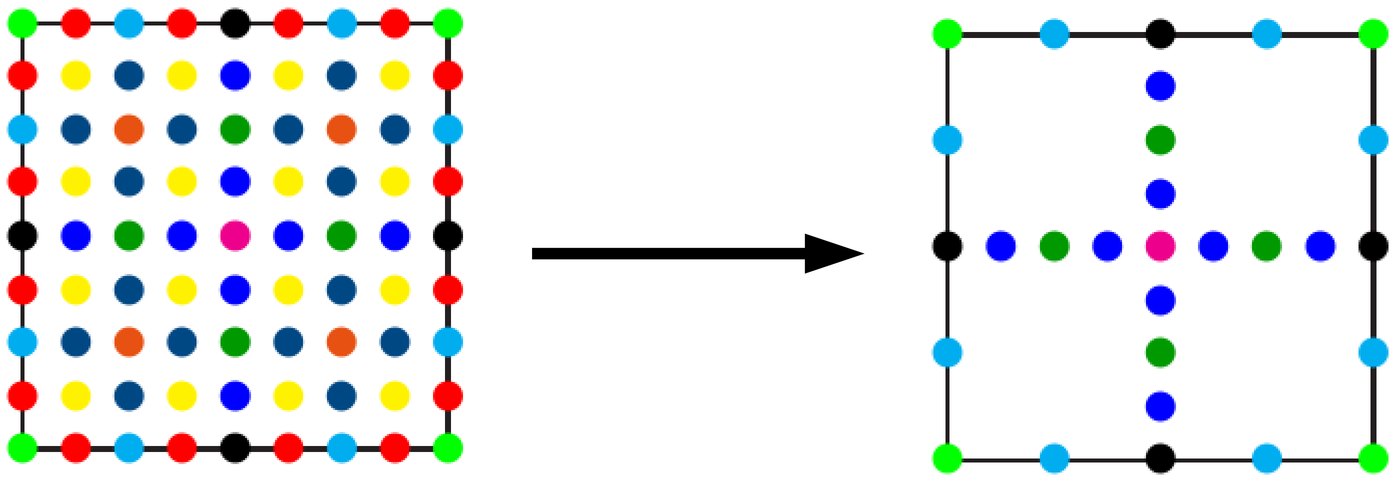
High-Performance Computing:

- Achieve massive speed-up by efficiently using modern supercomputers with hybrid parallelization scheme.

⇒ Parallel DP Adaptive-Sparse-Grid Algorithm

II. From Full Grids to Sparse Grids

(see, e.g. Zenger (1991), Bungartz & Griebel (2004), Garcke (2012), Pflüger (2010),...)



Interpolation on a Full Grid

-Consider a **1-dimensional function** $f : \Omega \rightarrow \mathbb{R}$ **on [0,1]**

-In numerical simulations:
f might be expensive to evaluate! (solve PDEs/system of non-linear Eqs.)
But: need to be able to evaluate *f* at arbitrary points using a numerical code

-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

- For simplicity: focus on case where $f|_{\partial\Omega} = 0$

Basis Functions

-Hierarchical basis based on **hat functions**

$$\phi(x) = \begin{cases} 1-|x| & \text{if } x \in [-1,1] \\ 0 & \text{else} \end{cases}$$

-Used to generate a **family of basis functions** $\phi_{l,i}$ having support $[x_{l,i} - h_l, x_{l,i} + h_l]$ by **dilation** and **translation**

$$\phi_{l,i}(x) := \phi\left(\frac{x - i \cdot h_l}{h_l}\right)$$

Hierarchical Increment Spaces

Hierarchical increment spaces:

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

with the **index set**

$$I_l = \{i \in \mathbb{N}, 1 \le i \le 2^l - 1, i \text{ odd}\}$$

The corresponding function space:

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

The **1d-interpolant:**

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

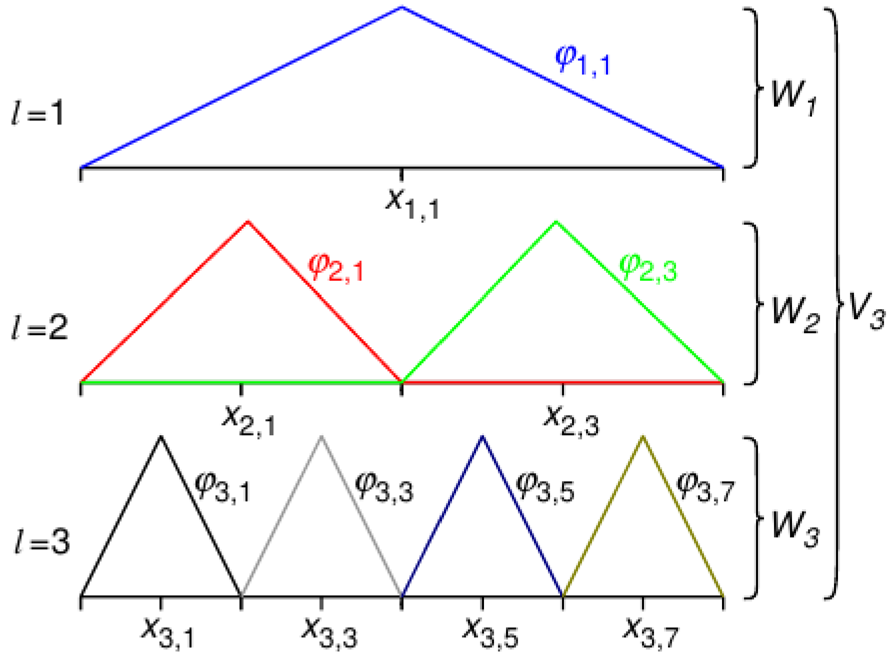


Fig.: **1-d basis functions** $\phi_{l,i}$ and the corresponding **grid points** up level ***l* = 3** in the hierarchical basis.

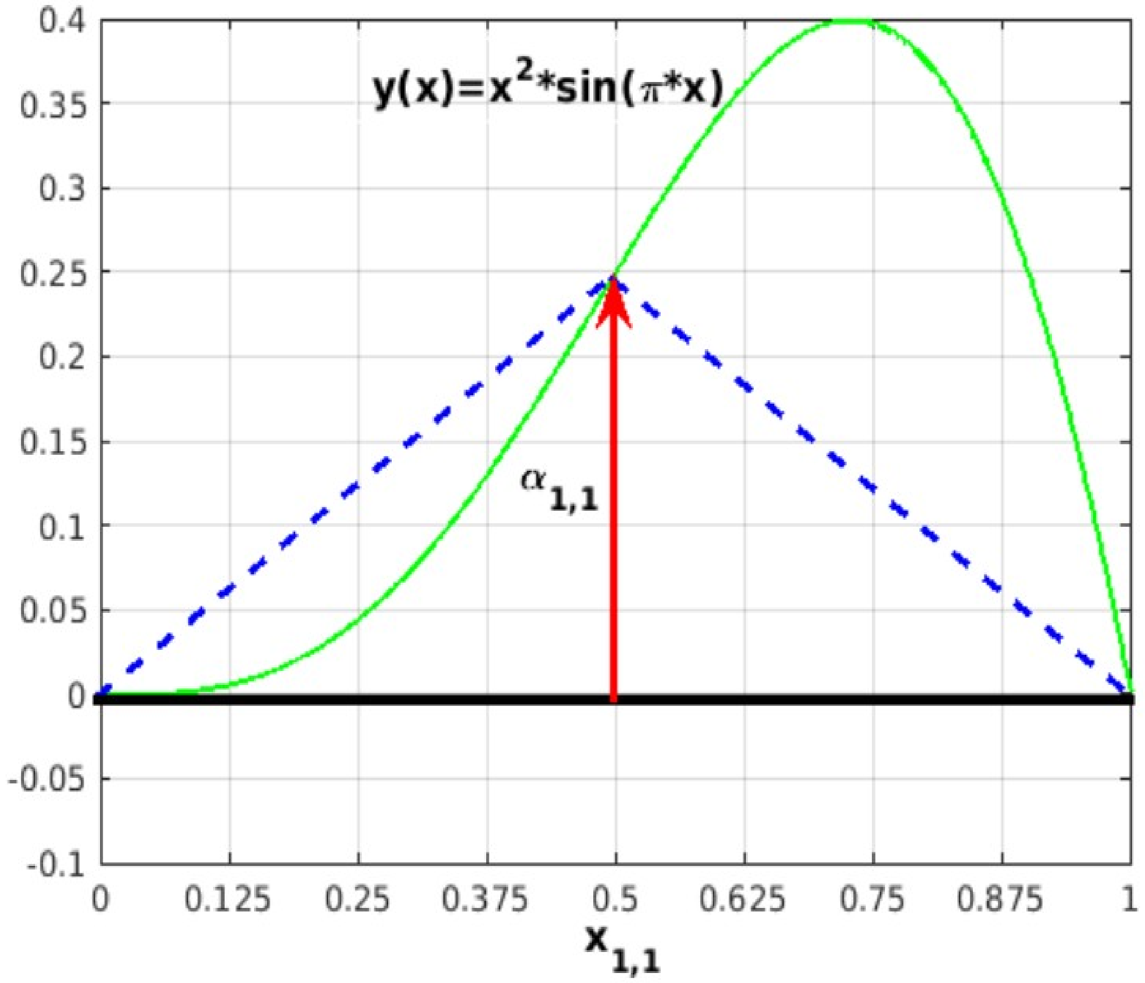
Note: supports of all basis functions of W_k mutually disjoint!

Piecewise Linear Interpolation: Level I

Coefficients:
hierarchical surpluses

They correct the interpolant of level *l*-1 at $\vec{x}_{l,i}$ to the actual value of $f(\vec{x}_{l,i})$

Nested structure:
Evaluate function only at points that are unique to the new level.

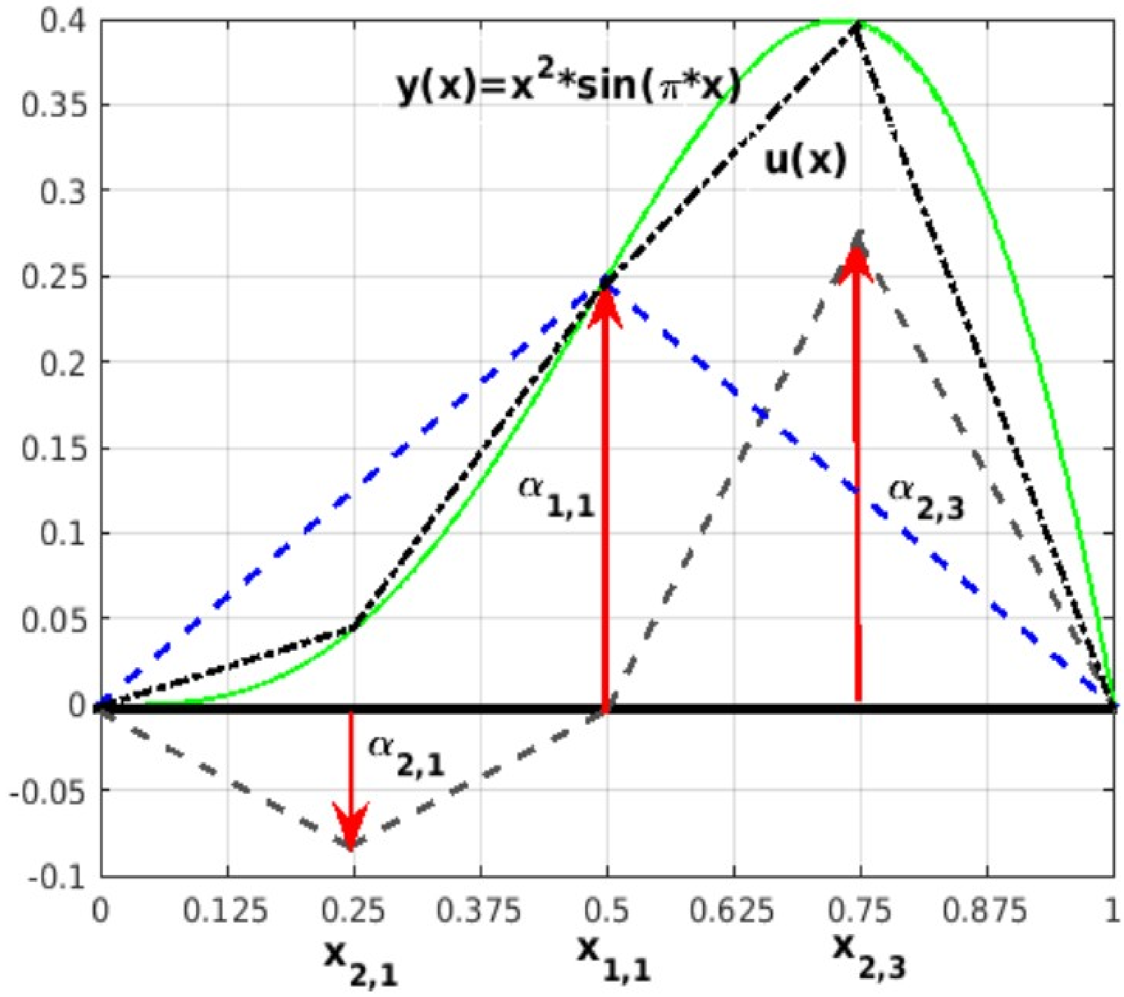


Piecewise Linear Interpolation: Level II

Coefficients:
hierarchical surpluses

They correct the interpolant of level *l*-1 at $\vec{x}_{l,i}$ to the actual value of $f\left(\vec{x}_{l,i}\right)$

Nested structure:
Evaluate function only at points that are unique to the new level.

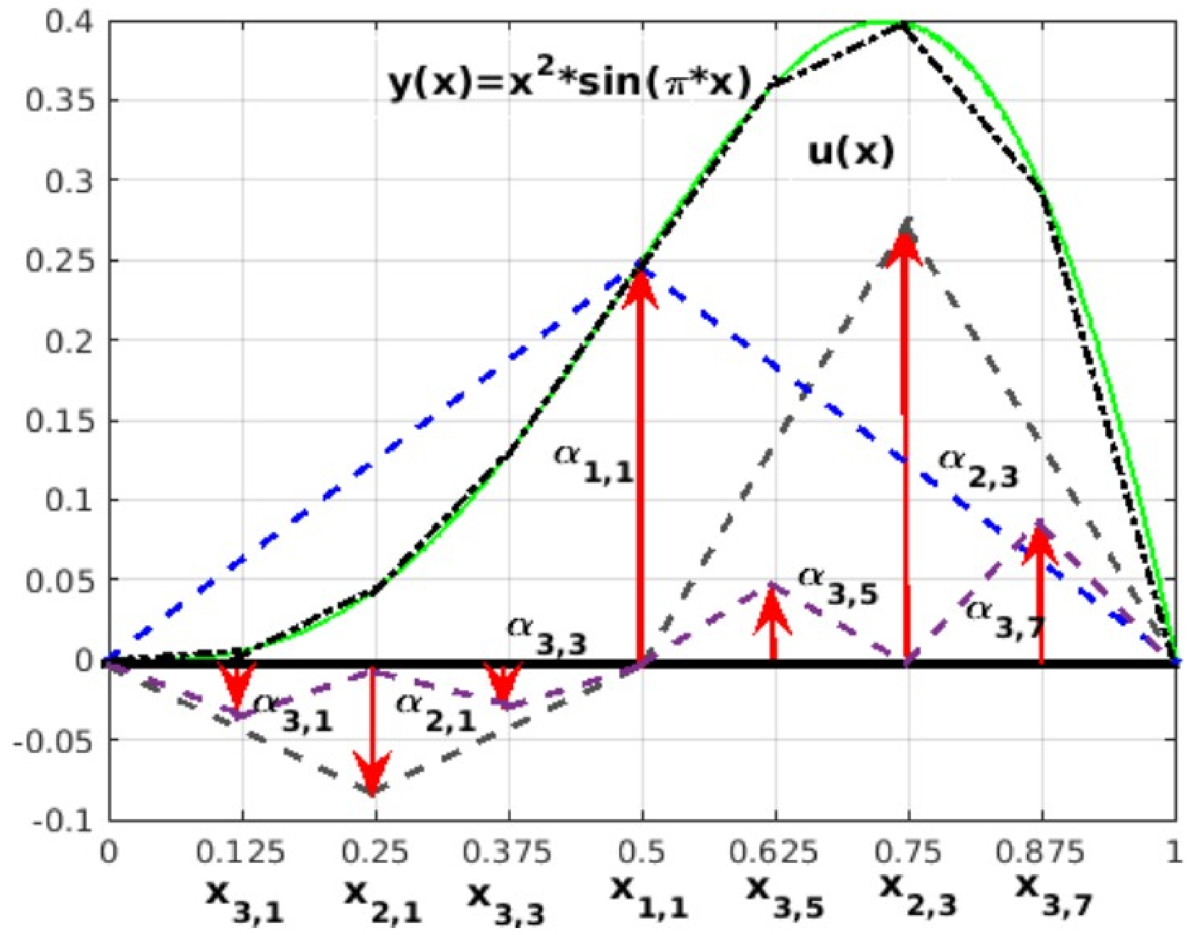


Piecewise Linear Interpolation: Level III

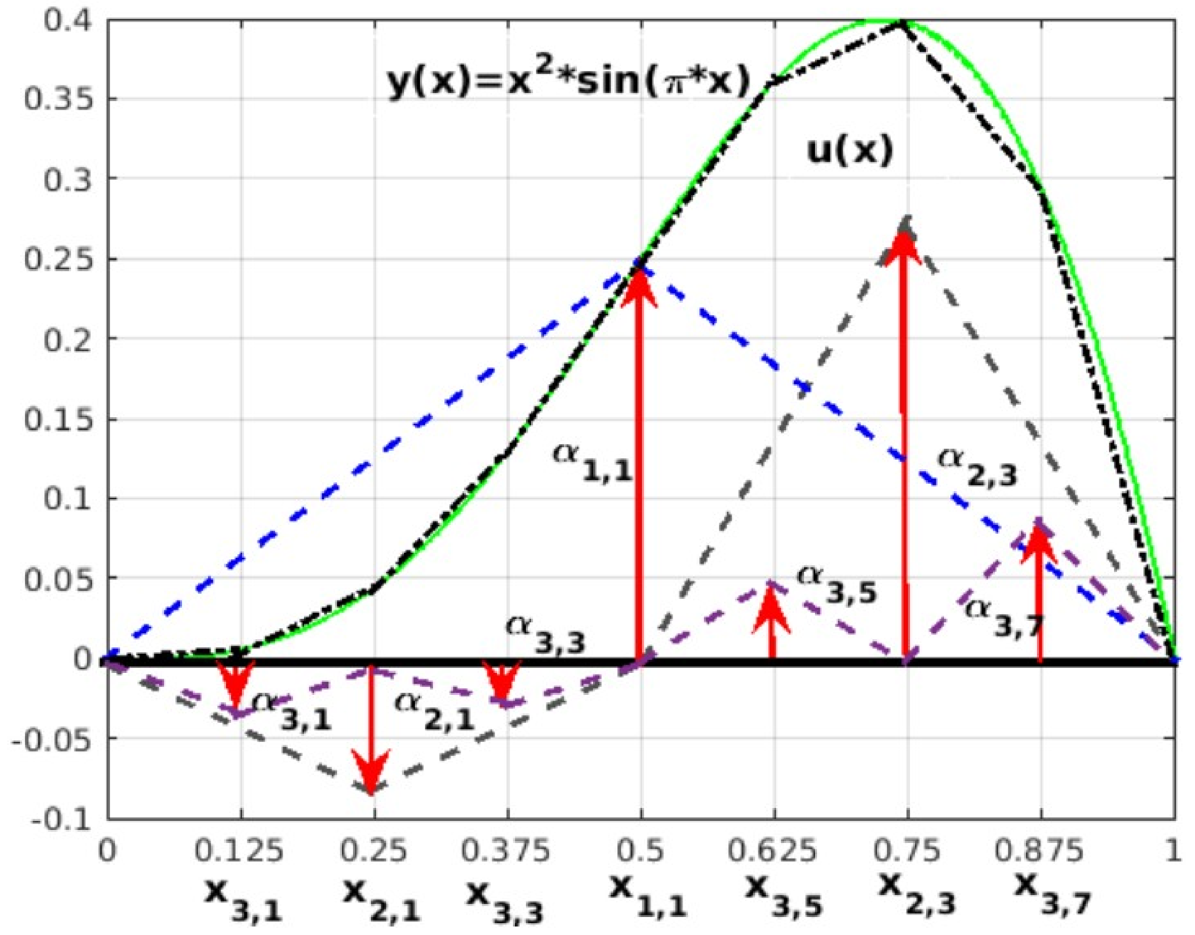
Coefficients:
hierarchical surpluses

They correct the interpolant of level *l*-1 at $\vec{x}_{l,i}$ to the actual value of $f\left(\vec{x}_{l,i}\right)$

Nested structure:
Evaluate function only at points that are unique to the new level.



MOVIE


y(x)=x^2*sin(π*x)
u(x)
α1,1
α2,3
α3,5
α3,7
α3,3
α3,1
α2,1
x3,1
x2,1
x3,3
x1,1
x3,5
x2,3
x3,7

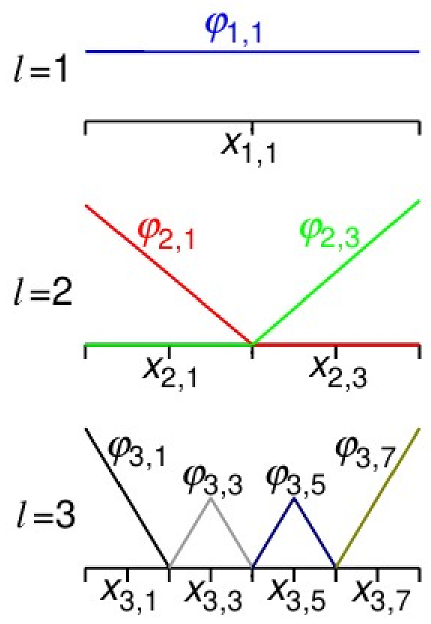
Non-zero Boundary Conditions

Want to be able to handle non-zero boundaries:

$$f|_{\partial\Omega} \neq 0$$

If we add naively points at boundaries, **3^d** support nodes will be added.

Numerically cheapest way:
Modify basis functions and interpolate towards boundary.
Various choices possible!



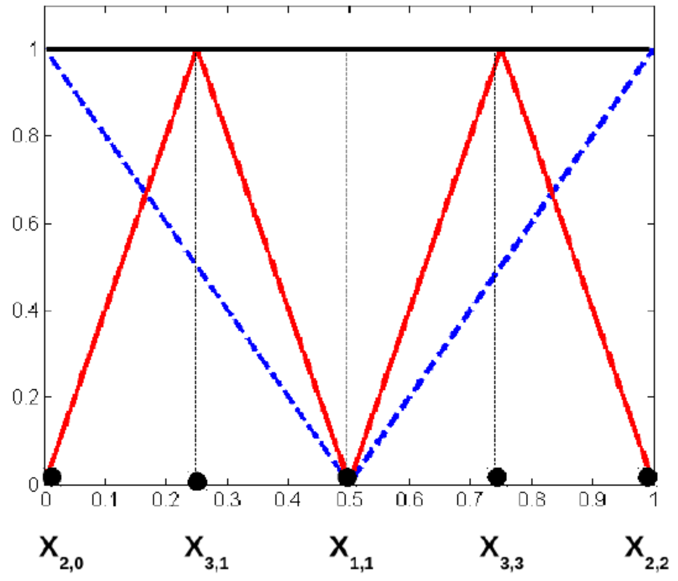


Fig.: Example of modified 1d-basis functions According to Pflüger (2010), which are extrapolating towards the boundary (**left**). They are constant on level 1 and **"folded-up"** if adjacent to the boundary on all other levels.
Right: "Modified" hat basis.

Some definitions & notation

(see, e.g. Zenger (1991), Bungartz & Griebel (2004), Garcke (2012), Pflüger (2010),...)

- We will focus on the domain **$\Omega = [0,1]^d$**

  d: dimensionality; **other domains: rescale**

- introduce **multi-indices**:

  grid refinement level: $\vec{l} = (l_1, ..., l_d) \in \mathbb{N}^d$

  spatial position: $\vec{i} = (i_1, ..., i_d) \in \mathbb{N}^d$

- Discrete, (Cartesian) full grid $\Omega_{\vec{l}}$ on Ω

- Grid $\Omega_{\vec{l}}$ consists of points: $\boxed{\vec{x}_{\vec{l},\vec{i}} := (x_{l_1,i_1}, ..., x_{l_d,i_d})}$

  Where $x_{l_t,i_t} := i_t \cdot h_{l_t} = i_t \cdot 2^{-l_t}$ and $i_t \in \{0, 1, ..., 2^{l_t}\}$

Multi-Dimensional Interpolant

Extension to multi-d by a **tensor-product construction**:

Multi-d basis: $\phi_{\vec{l},\vec{i}}(\vec{x}) := \prod_{t=1}^{d} \phi_{l_t,i_t}(x_t)$

Index set: $I_{\vec{l}} := \{\vec{i} : 1 \leq i_t \leq 2^{l_t} - 1, i_t \text{ odd}, 1 \leq t \leq d\}$

Hierarchical increments: $W_{\vec{l}} := \text{span}\{\phi_{\vec{l},\vec{i}} : \vec{i} \in I_{\vec{l}}\}$

Multi-d interpolant:

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

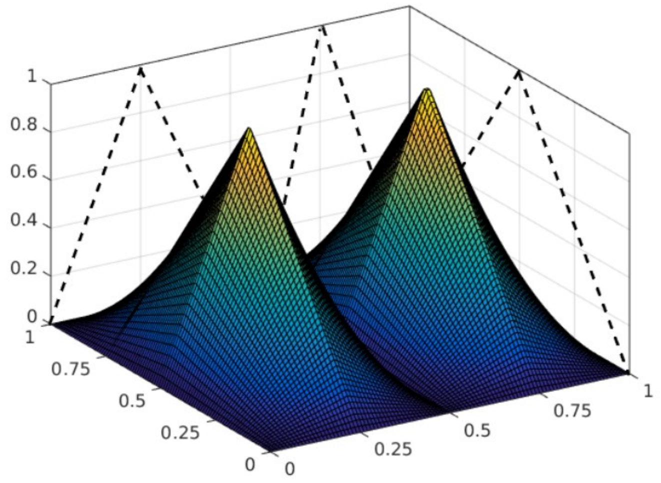


Fig.: Basis functions of the **subspace $W_{2,1}$**

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 I

Question: **"can we construct discrete approximation spaces that are better in the sense that the same number of invested grid points leads to a higher order of accuracy?"** **YES √**

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

→ If **second mixed derivatives are bounded**, then the hierarchical **surpluses decay rapidly** with increasing approximation level.

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

`Breaking' the curse of dimensionality II

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

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

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

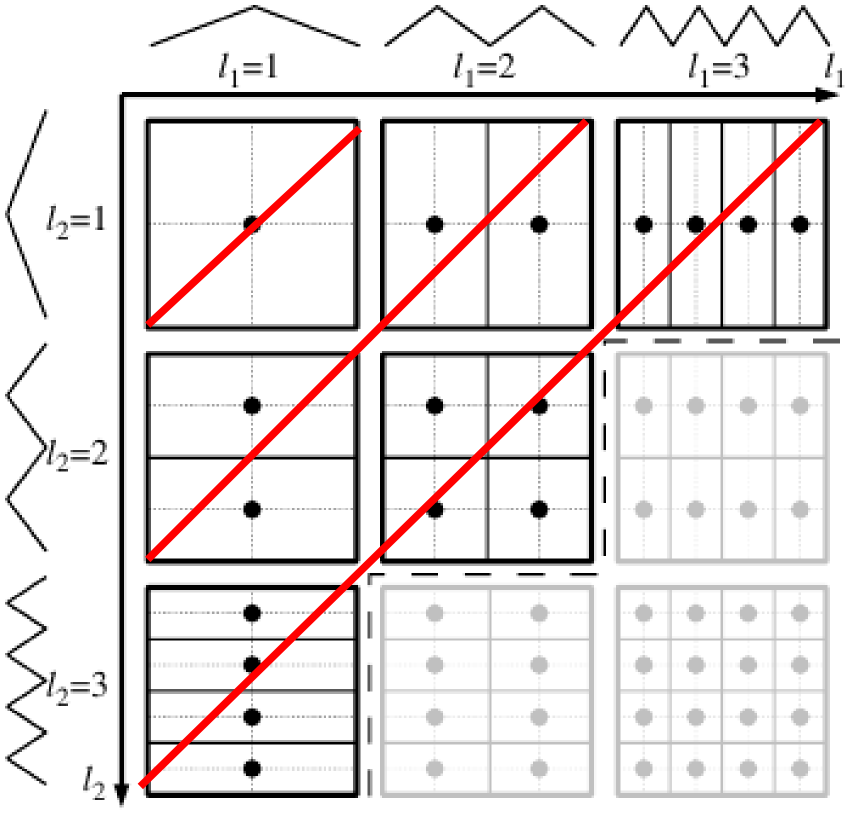
$$V_{0,n}^S := \bigoplus_{|\vec{l}|_1 \leq n+d-1} W_{\vec{l}}.$$

Interpolant: $f_{0,n}^S(\vec{x}) \approx u(\vec{x}) = \sum_{|l|_1 \leq n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot \phi_{\vec{l},\vec{i}}(\vec{x})$

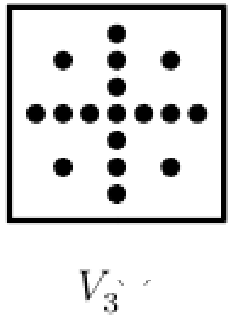
# grid points: $\underline{\mathcal{O}\left(h_n^{-1} \cdot (\log(h_n^{-1}))^{d-1}\right) = \mathcal{O}\left(2^n \cdot n^{d-1}\right)}$ << $\mathcal{O}\left(h_n^{-d}\right) = \mathcal{O}\left(2^{nd}\right)$

Accuracy of the interpolant: $\underline{\mathcal{O}\left(h_n^2 \cdot \log(h_n^{-1})^{d-1}\right)}$ vs. $\mathcal{O}\left(h_n^2\right)$

Sparse grid construction in 2D



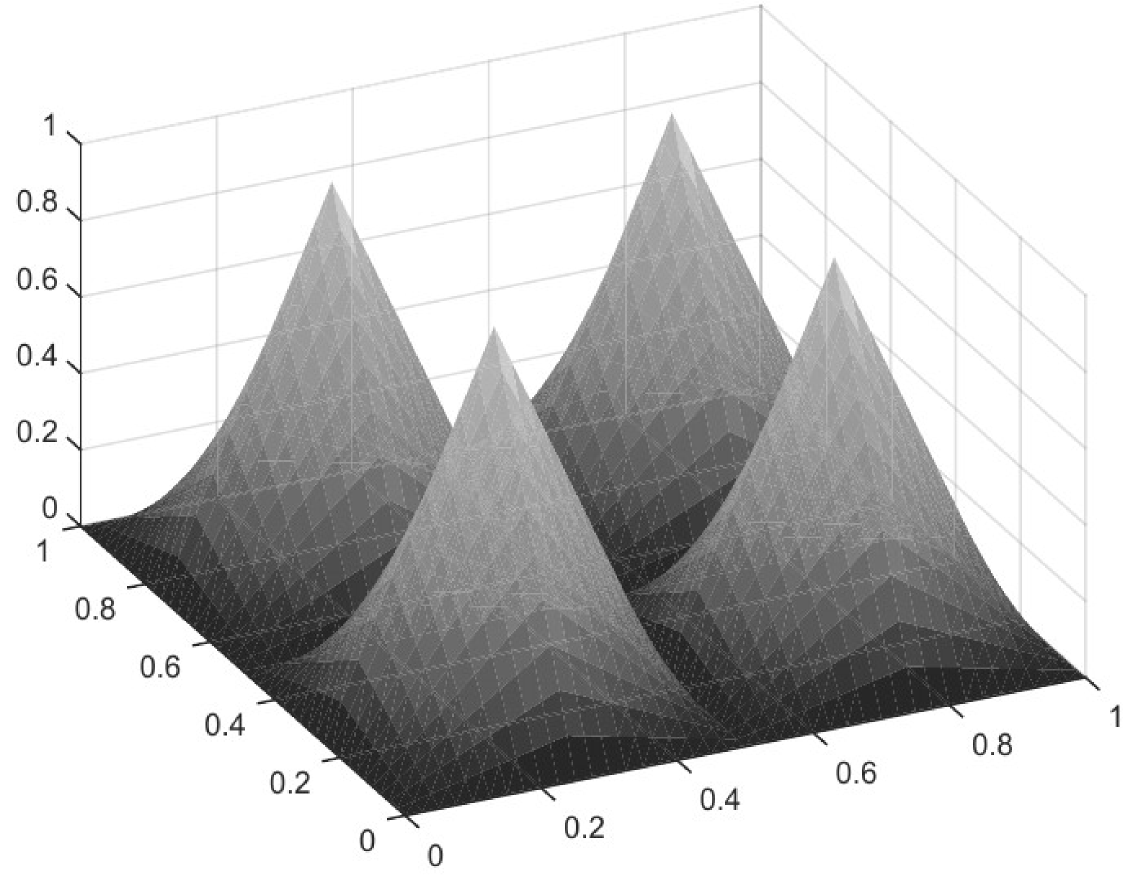
$$V_{0,n}^{S} := \bigoplus_{|\vec{l}|_1 \le n+d-1} W_{\vec{l}}$$



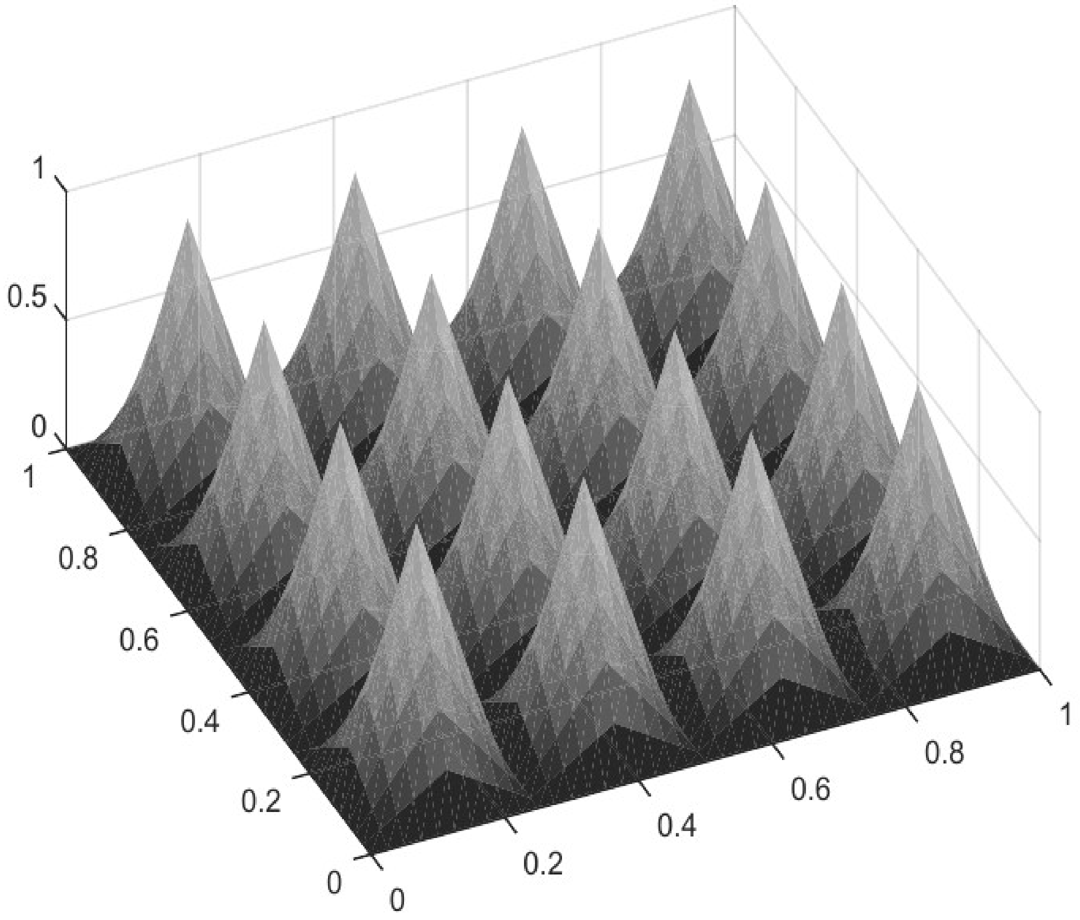
Sparse grid: 17 pt.
Full grid : 49 pt.

Fig.: Two-dimensional subspaces **W$_l$** up to ***l***=3 (h$_3$ = 1/8) in each dimension.
The optimal a priori selection of subspaces in shown in black **(left)** and the Corresponding sparse grid of level n = 3 **(right)**.
For the full grid, the gray subspaces have to be used as well.

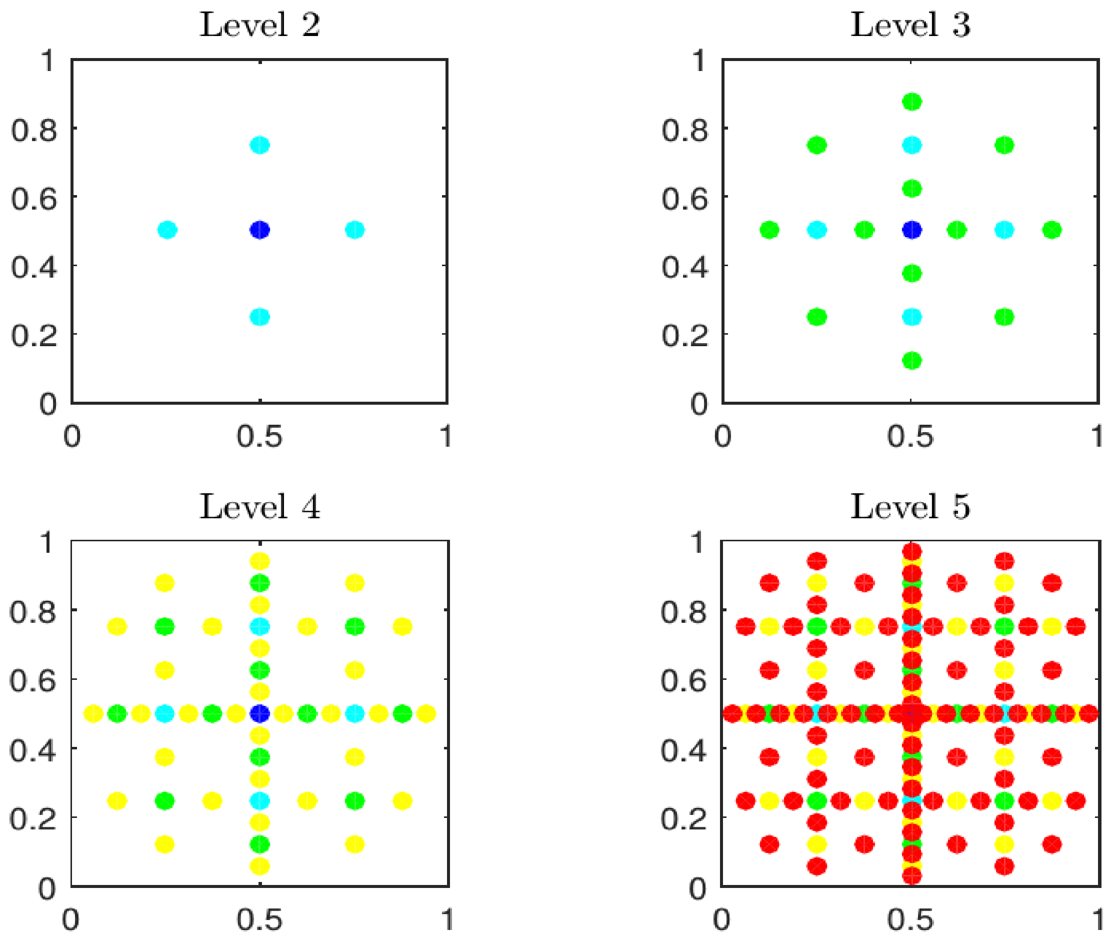
Basis Functions of $W_{2,2}$ — Included in V_3



Basis Functions of $W_{3,3}$ — not Included in V_3

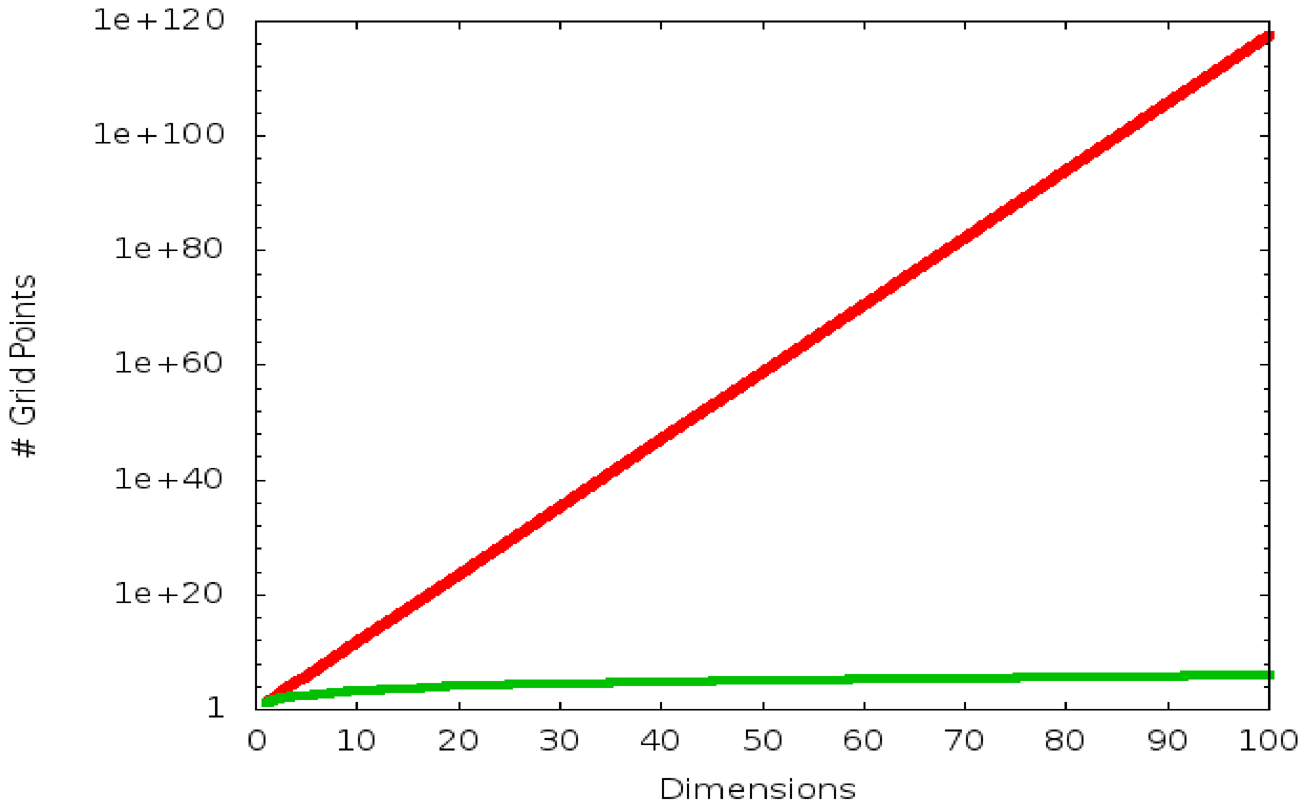


Sparse Grid of Increasing level



Grid Points

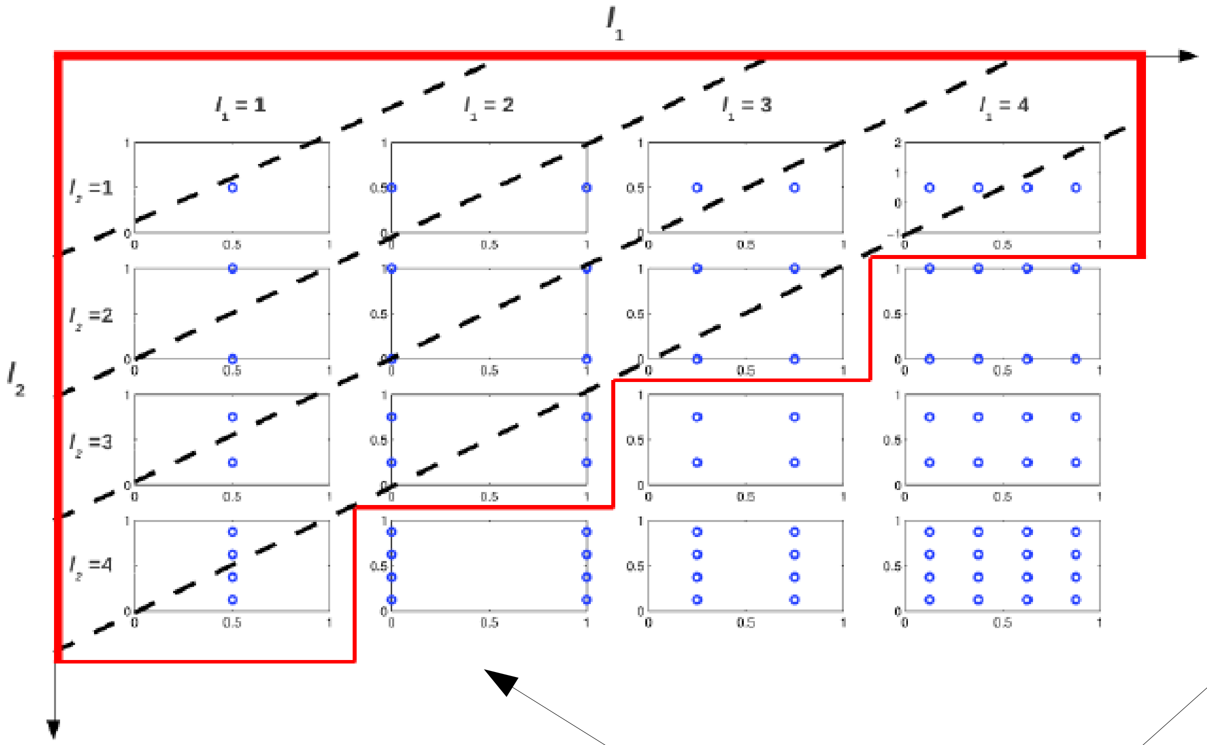
d	$\lvert V_n \rvert$	$\lvert V_{0,n}^S \rvert$
1	15	15
2	225	49
3	3375	111
4	50'625	209
5	759'375	351
10	$5.77 \cdot 10^{11}$	2'001
15	$4.37 \cdot 10^{17}$	5'951
20	$3.33 \cdot 10^{23}$	13'201
30	$1.92 \cdot 10^{35}$	41'601
40	$1.11 \cdot 10^{47}$	95'201
50	$6.38 \cdot 10^{58}$	182'001
100	>Googol	1'394'001

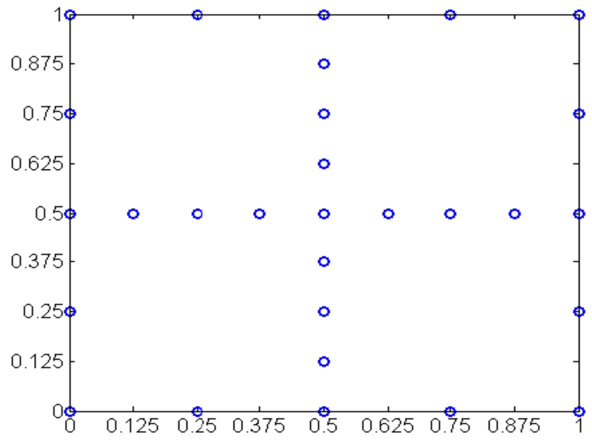


Tab.: Number of grid points for several types of sparse grids of level *n* = 4.
Middle: Full grid; **right:** **classical sparse grid** with **no points at the boundaries.**
Fig.: Number of grid points growing with dimension (full grid vs. sparse grid).

Sparse Grid with non-zero boundaries

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





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

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

Hierarchical Integration

High-dimensional integration easy with sparse grids, e.g. compute expectations
Let's assume uniform probability density:

$$\mathbb{E}\left[u(\vec{x})\right] = \sum_{|l|_1 \le n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \int_\Omega \phi_{\vec{l},\vec{i}}(\vec{x}) d\vec{x}$$

The one-dimensional integral can now be computed analytically (Ma & Zabras (2008))

$$\int_0^1 \phi_{l,i}(x)\, dx = \begin{cases} 1, & \text{if } l = 1 \\ \frac{1}{4} & \text{if } l = 2 \\ 2^{1-l} & \text{else} \end{cases}$$

Note that this result is independent of the location of the interpolant to dilation
And translation properties of the hierarchical basis functions.
→ **Multi-d integrals are therefore again products of 1-d integrals.**

We denote $\int_\Omega \phi_{l,i}(\vec{x})\, d\vec{x} = J_{\vec{l},\vec{i}}$

$$\longrightarrow \mathbb{E}\left[u(\vec{x})\right] = \sum_{|l|_1 \le n+d-1} \sum_{\vec{i} \in I_{\vec{l}}} \alpha_{\vec{l},\vec{i}} \cdot J_{\vec{l},\vec{i}}$$

Where are Sparse Grids used?

For a review, see, e.g. Bungartz & Griebel (2004)

Sparse grid methods date back to Smolyak(1963)

BUT: Smolyak used global polynomials!

So far, methods applied to:

-High-dimensional integration

e.g. Gerstner & Griebel (1998), Bungartz et al. (2003),...

-Interpolation

e.g. Barthelmann et al. (2000), Klimke & Wohlmut (2005),...

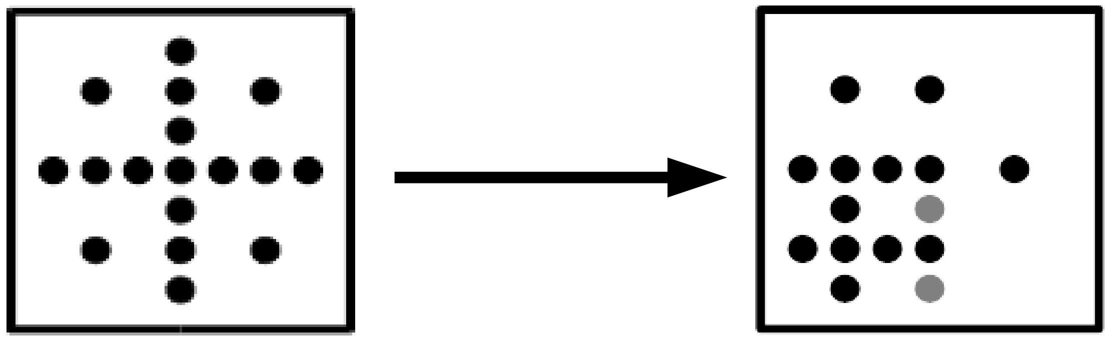
-Solution of PDEs

e.g. Zenger (1991), Griebel (1998),...

More fields of application: regressions, data mining, likelihood estimations, option pricing, data compression, dynamic economic models...

e.g. Kubler & Kruger (2004), Winschel & Kraetzig (2010), Judd et al. (2013) → Smolyak; global basis functions.

III. Adaptive Sparse Grids



Sketch of adaptive refinement

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

-Surpluses should quickly decay to zero

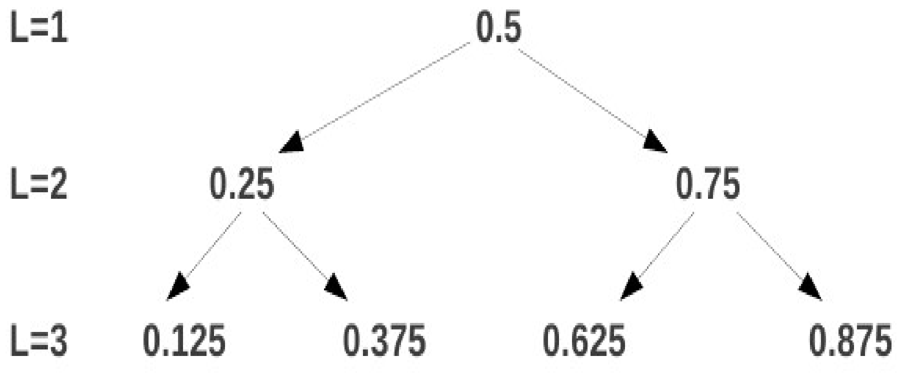
-**Use hierarchical surplus as error indicator**.

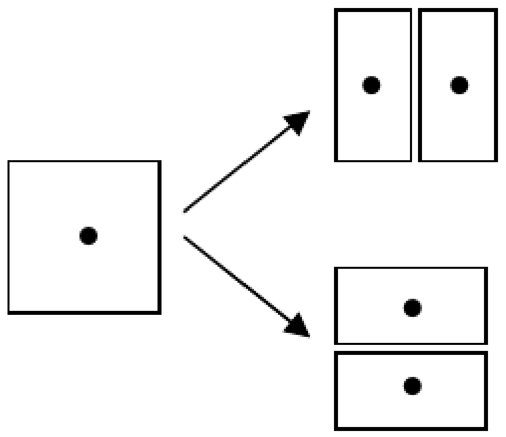
-**Automatically detect "discontinuity regions"** and adaptively refine the points in this region.

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

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

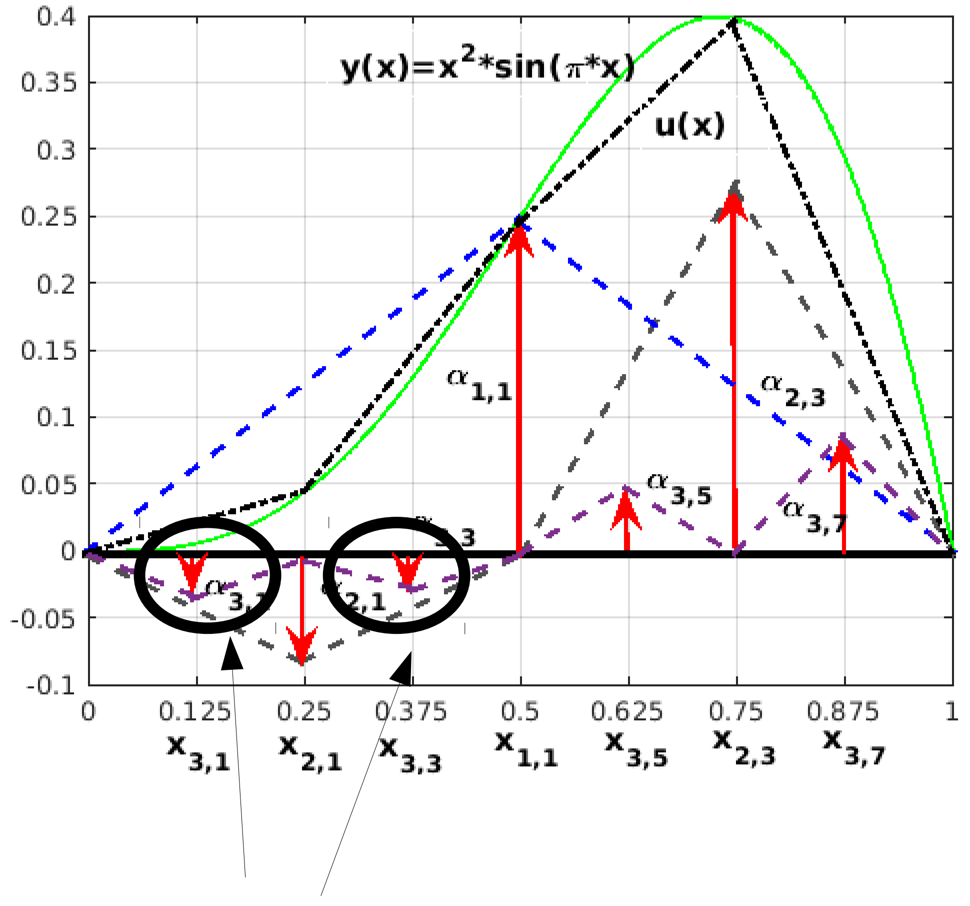
-Criterion: **e.g.** $|\alpha_{\vec{l},\vec{i}}| \geq \epsilon$





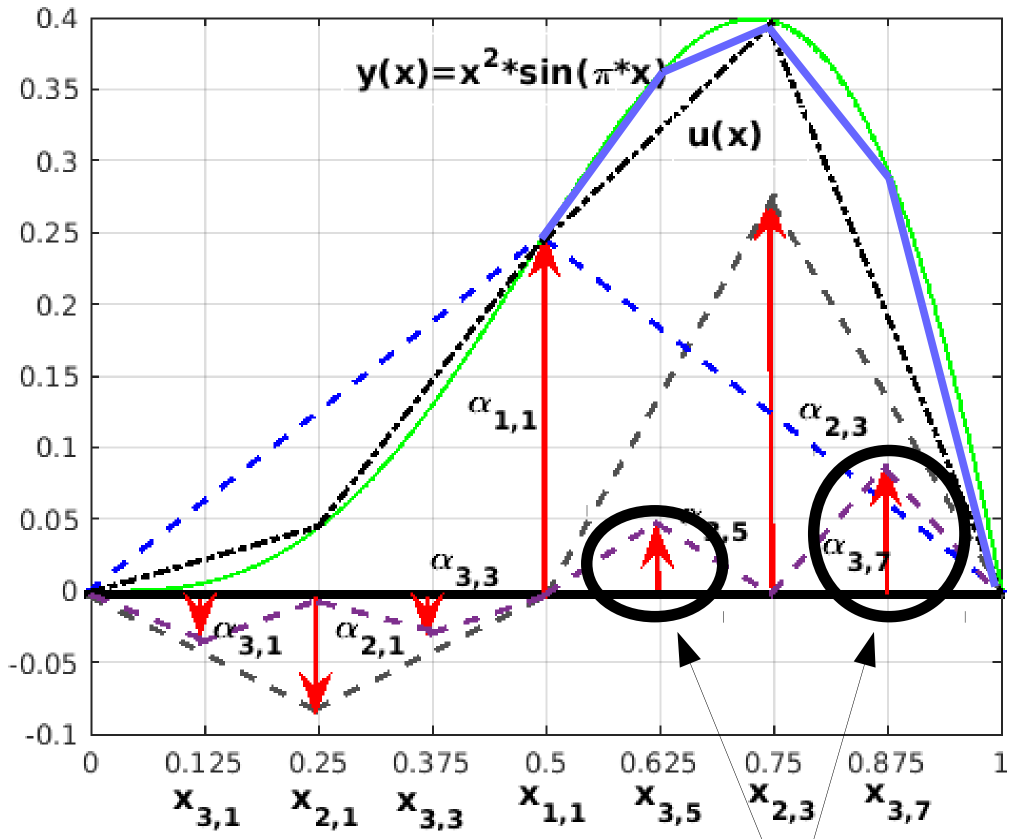
top panel: tree-like structure of sparse grid.
lower panel: locally refined sparse grid in 2D.

Example I



Small – below threshold

Example II



Add points – above threshold

Test in 1d
(See Genz (1984) for test functions)

Test function:

$$f(x) = \frac{1}{|0.5 - x^4| + 0.01}$$

Error both for full grid and adapt. sparse grid of $O(10^{-2})$.

Error measure:
→ 1000 random points from [0,1]

$$e = \max_{i=1,\ldots,1000} |f(\vec{x}_i) - u(\vec{x}_i)|$$

Full grid: **1023** points
Adaptive sparse grid: **109** points.

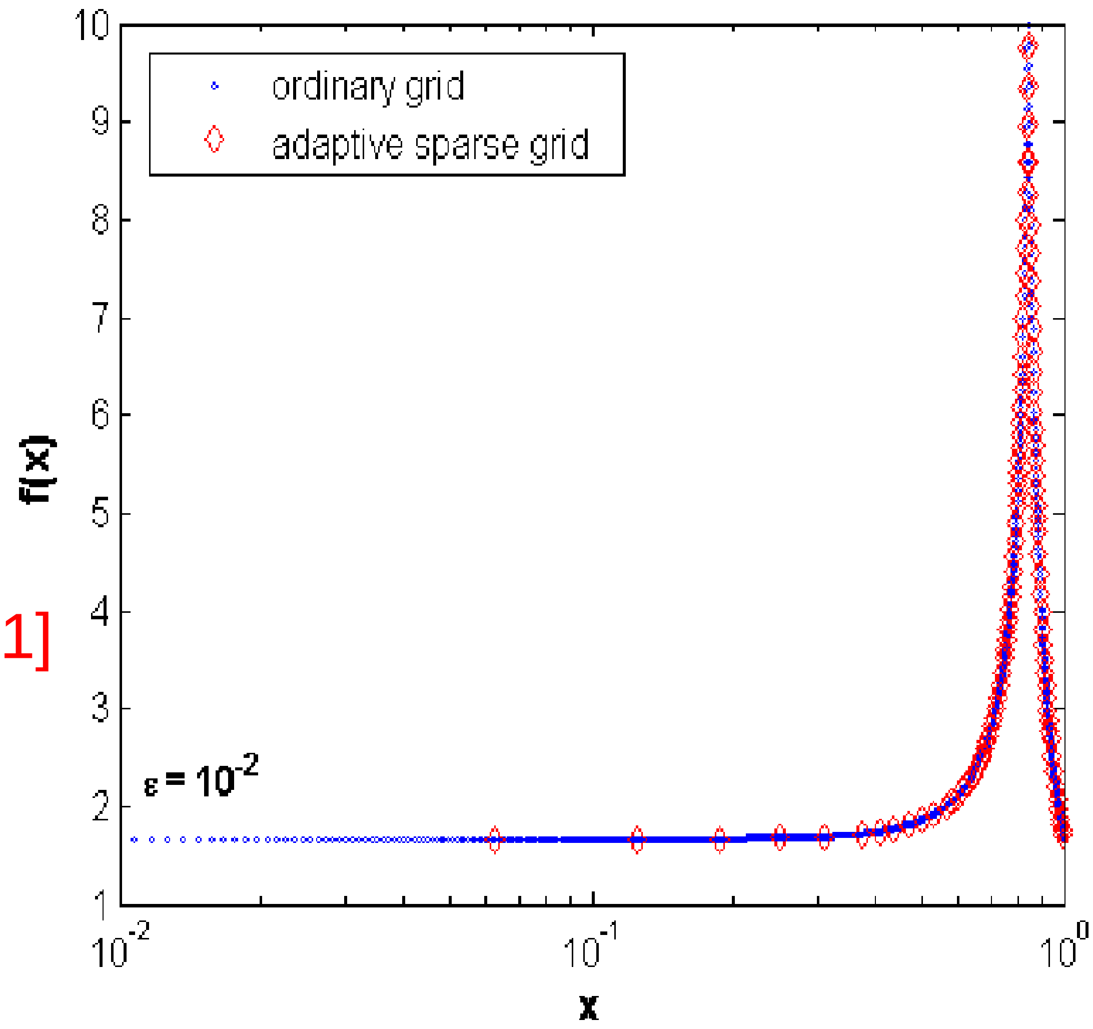


Fig.: Blue: Full grid;red: adaptive sparse grid.

Test in 2d

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

Error: *$O(10^{-2})$*

Full grid:
→ *$O(10^{9})$* points

Sparse grid:
→ **311,297 points**

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

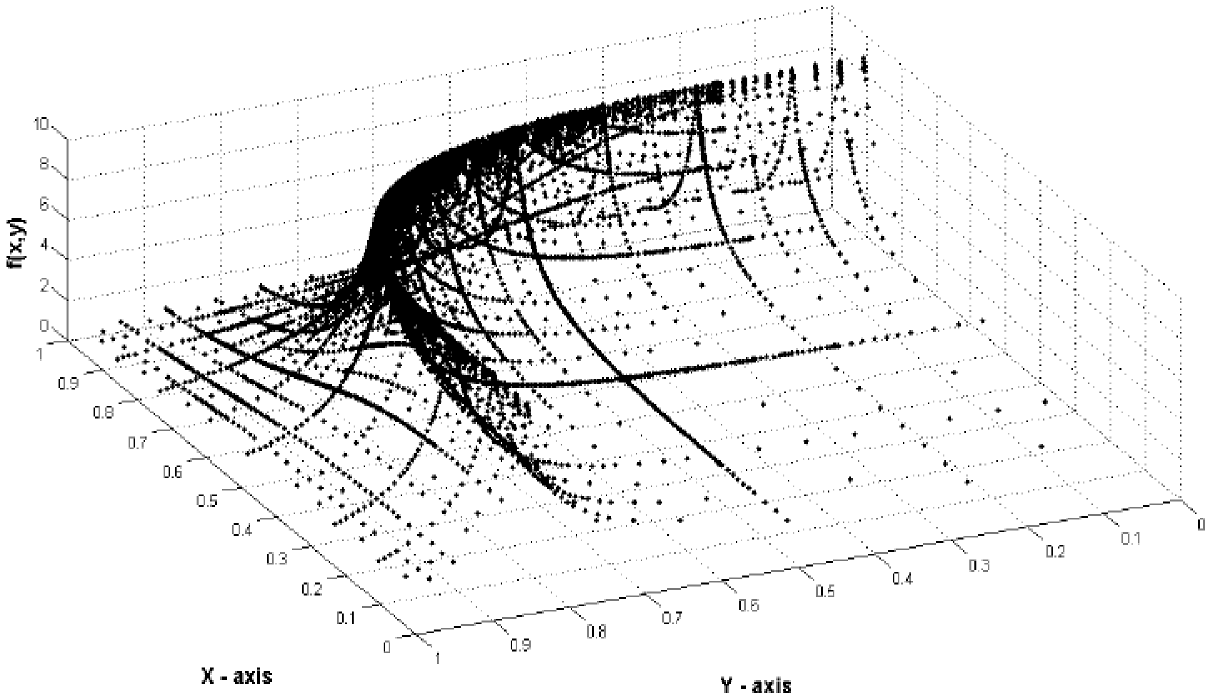


Fig.: 2d test function and its corresponding grid points after 15 refinement steps.

Movie

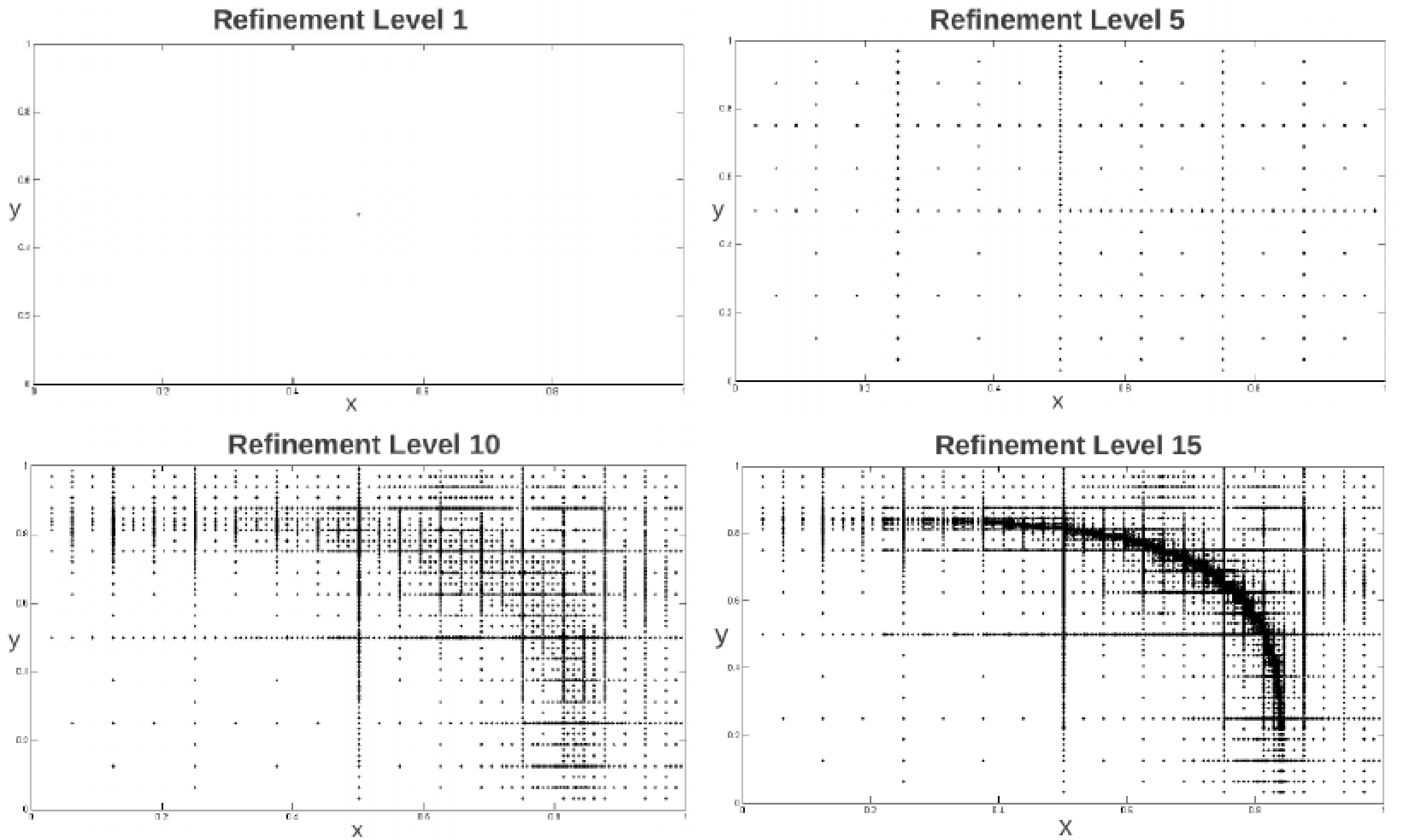
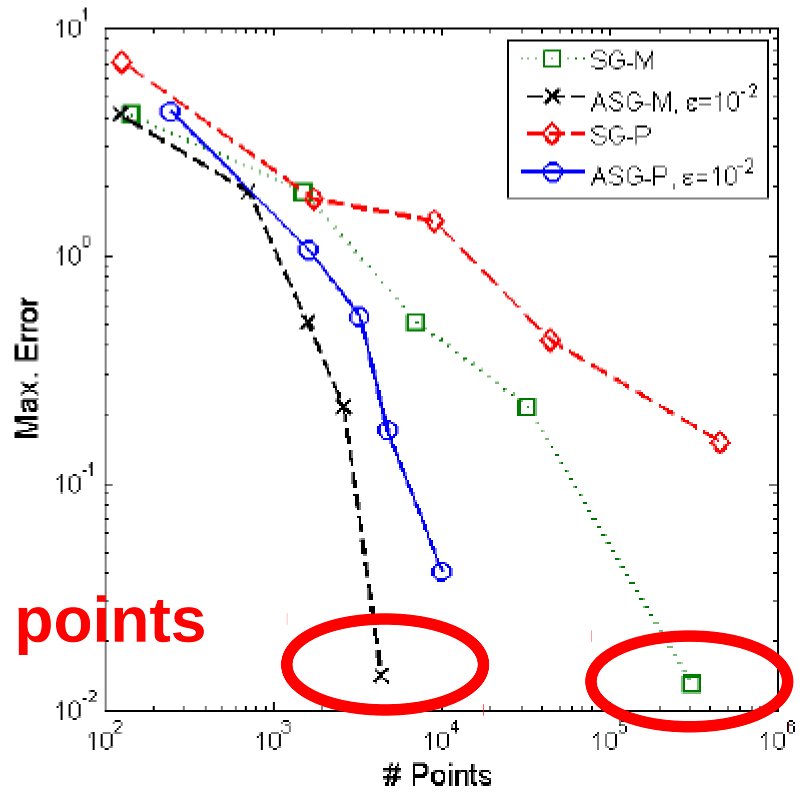


Fig.: Evolution of the adaptive sparse grid with a **threshold for refinement of 10^{-2}** . The refinement levels displayed are L = 1, 5, 10, 15.

Convergence

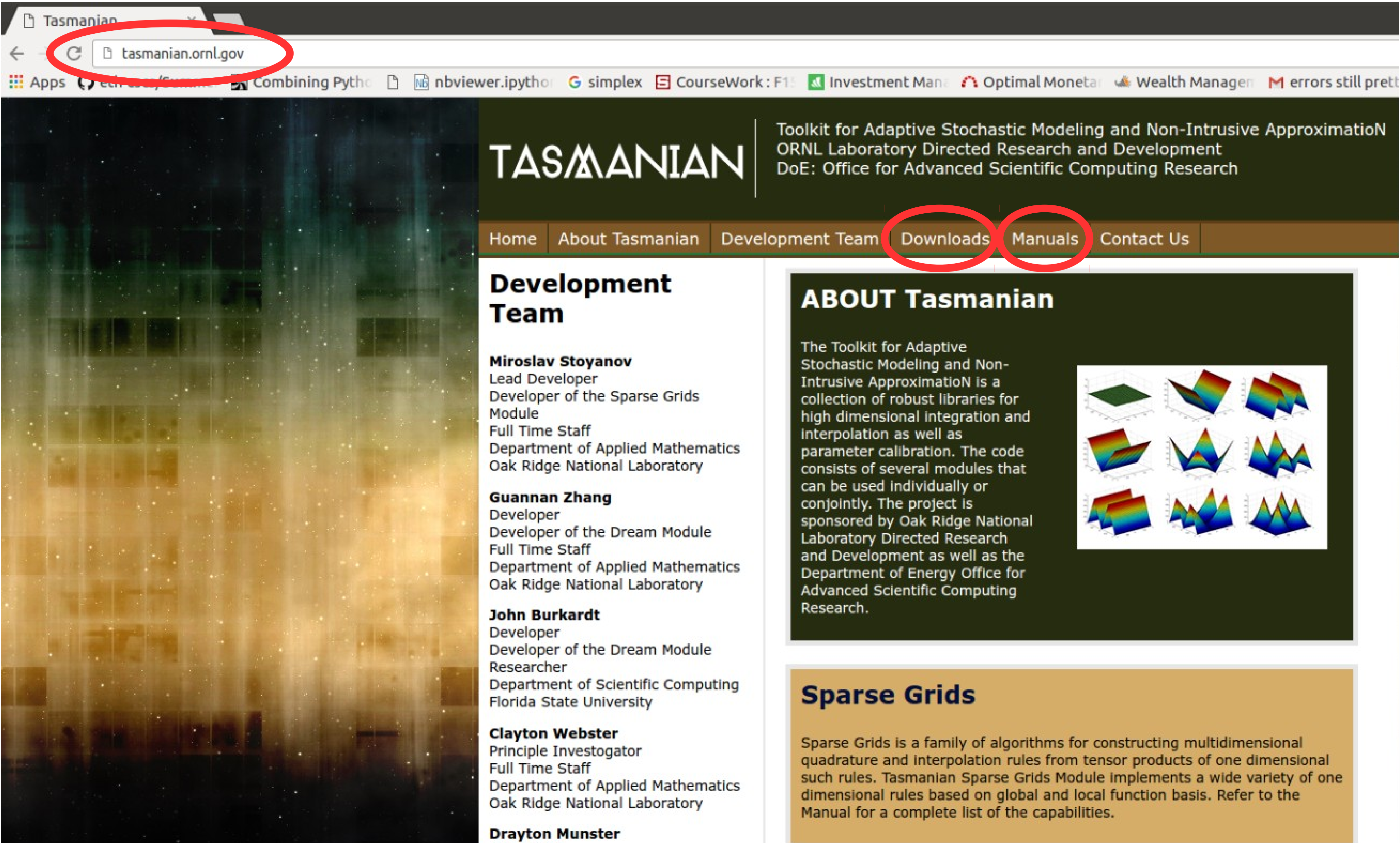


4,411 points

311,297 points

Fig.: Comparison of the interpolation error for **conventional and adaptive sparse grid interpolation** (two different adaptive sparse grid choices).

TASMANIAN – open source ASG



Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN
ORNL Laboratory Directed Research and Development
DoE: Office for Advanced Scientific Computing Research

Development Team

Miroslav Stoyanov
Lead Developer
Developer of the Sparse Grids Module
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

Guannan Zhang
Developer
Developer of the Dream Module
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

John Burkardt
Developer
Developer of the Dream Module
Researcher
Department of Scientific Computing
Florida State University

Clayton Webster
Principle Investogator
Full Time Staff
Department of Applied Mathematics
Oak Ridge National Laboratory

Drayton Munster

ABOUT Tasmanian

The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN is a collection of robust libraries for high dimensional integration and interpolation as well as parameter calibration. The code consists of several modules that can be used individually or conjointly. The project is sponsored by Oak Ridge National Laboratory Directed Research and Development as well as the Department of Energy Office for Advanced Scientific Computing Research.

Sparse Grids

Sparse Grids is a family of algorithms for constructing multidimensional quadrature and interpolation rules from tensor products of one dimensional such rules. Tasmanian Sparse Grids Module implements a wide variety of one dimensional rules based on global and local function basis. Refer to the Manual for a complete list of the capabilities.

Software tutorial in the afternoon!!

The Toolkit for Adaptive Stochastic Modeling and Non-Intrusive ApproximatioN

http://tasmanian.ornl.gov/

TASMANIAN Sparse Grids v3.1 (February 2016).

Very recent open source library written in CPP:

→ Contains “ordinary and adaptive” sparse grids.

→ Many more basis functions (global polynomials, wavelets,…).

→ Interfaces to Python and Matlab.

→ Moderately parallelized (with OpenMP).

Compile & run TASMANIAN*

!!! READ THE * MANUAL (RTFM) – p.50ff !!!**

1. go to TASMANIAN:

```
> cd zice17/simon/codes/TasmanianSparseGrids
```

2. compile:

```
> make
```

3. go to simple example:

```
> cd TASMANIAN/TasmanianSparseGrids/ZICE17_Matlab
```

4. let's have a look at the example:

```
> tsg_example_ZICE17.m
```

5. launch matlab & run example:

```
> matlab -nojvm  (no gui)
> tsg_example_ZICE17()
```

6. NOTE: Tasmanian [-1,1]^d instead of [0,1]^d

If you are interested in CPP code examples, TASMANIAN provides examples here:

```
> cd TASMANIAN/TasmanianSparseGrids/Example/example.cpp
> make
```

*more examples provided from TASMANIAN in InterfaceMATLAB/tsgExample.m → beyond scope of lecture.

Exercises

Create sparse grids based on different analytical test functions, e.g. Genz (1984).

→ different test functions can be obtained by varying $c = (c_1, \ldots, c_d)$ (c> 0) and $w = (w_1, \ldots, w_d)$

→ difficulty of functions is monotonically increasing with c.

→ randomly generate 1,000 test points and compute error(s): $e = \max_{i=1,\ldots,1000} |f(\vec{x}_i) - u(\vec{x}_i)|.$

→ **play with adaptive/non-adaptive sparse grids/refinement level and criterion.**

→ generate convergence plots (number of points versus error – as done above).

Genz (1984) test functions

1. OSCILLATORY: $f_1(x) = \cos\left(2\pi w_1 + \sum_{i=1}^{d} c_i x_i\right),$
2. PRODUCT PEAK: $f_2(x) = \prod_{i=1}^{d} \left(c_i^{-2} + (x_i - w_i)^2\right)^{-1},$
3. CORNER PEAK: $f_3(x) = \left(1 + \sum_{i=1}^{d} c_i x_i\right)^{-(d+1)},$
4. GAUSSIAN: $f_4(x) = \exp\left(-\sum_{i=1}^{d} c_i^2 t(x_i - w_i)^2\right),$
5. CONTINUOUS: $f_5(x) = \exp\left(-\sum_{i=1}^{d} c_i |x_i - w_i|\right),$
6. DISCONTINUOUS: $f_6(x) = \begin{cases} 0, & \text{if } x_1 > w_1 \text{ or } x_2 > w_2, \\ \exp\left(\sum_{i=1}^{d} c_i x_i\right), & \text{otherwise.} \end{cases}$

1. III. Putting things together: Iterative schemes*, Adaptive Sparse Grids, HPC

*method applies both to dynamic programming as well as to time iteration

IRBC with Adjustment Costs and Irreversible Investment

- Use test case from JEDC project on computational methods
(Den Haan et al. 2011, Juillard and Villemot 2011, Kollmann et al. 2011)

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

⇒ countries differ in productivity, ***a*** (stochastic and exogen.), and capital stock, ***k*** (endogen.)
⇒ dimension of the state space is ***2N***
⇒ recursive equilibrium is characterized by
policy $p: R_+^{2N} \Rightarrow R_+^{N+1}$
mapping state into ***N*** capital choices and ***1*** Lagrange multiplier (as markets are complete)

- Extension: Investment in each country is irreversible:

⇒ recursive equilibrium is characterized by policy
$p: R_+^{2N} \Rightarrow R_+^{2N+1}$
mapping state into ***N*** capital choices, ***1*** Lagrange multiplier,
and ***N*** Karush-Kuhn-Tucker multipliers

Time Iteration Algorithm: Detail

N Euler equations

$$g^j_{t+1} := k^j_{t+1}/k^j_t - 1,\;\; g^j_{t+2} := k^j_{t+2}/k^j_{t+1} - 1,$$

$$\forall j : \lambda_t \cdot \left[1 + \phi \cdot g^j_{t+1}\right] - \beta \cdot \mathbb{E}_t \left\{ \lambda_{t+1} \cdot \left[a^j_{t+1} \cdot A \cdot \alpha \cdot (k^j_{t+1})^{\alpha - 1} + (1-\delta) + \frac{\phi}{2} \cdot g^j_{t+2} \cdot \left(g^j_{t+2} + 2 \right) \right] \right\} = 0,$$

$$\sum_{j=1}^{N} \left(a^j_t \cdot A \cdot (k^j_t)^\alpha + k^j_t \cdot \left((1-\delta) - \frac{\phi}{2} \cdot (g^j_{t+1})^2 \right) - k^j_{t+1} - \left(\frac{\lambda_t}{\tau_j} \right)^{-\gamma^j} \right) = 0.$$

1 aggregate resource constraint

We solve the model by iterating on these FOCs, i.e. at each grid point

$$(a^1_t, \ldots, a^N_t, k^1_t, \ldots, k^N_t) \longrightarrow$$ **Current grid points**

solve for the unknown policy variables

$$(k^1_{t+1}, \ldots, k^N_{t+1}, \lambda_t) \longrightarrow$$ **Solution of system of Eqs.**
At the current iteration

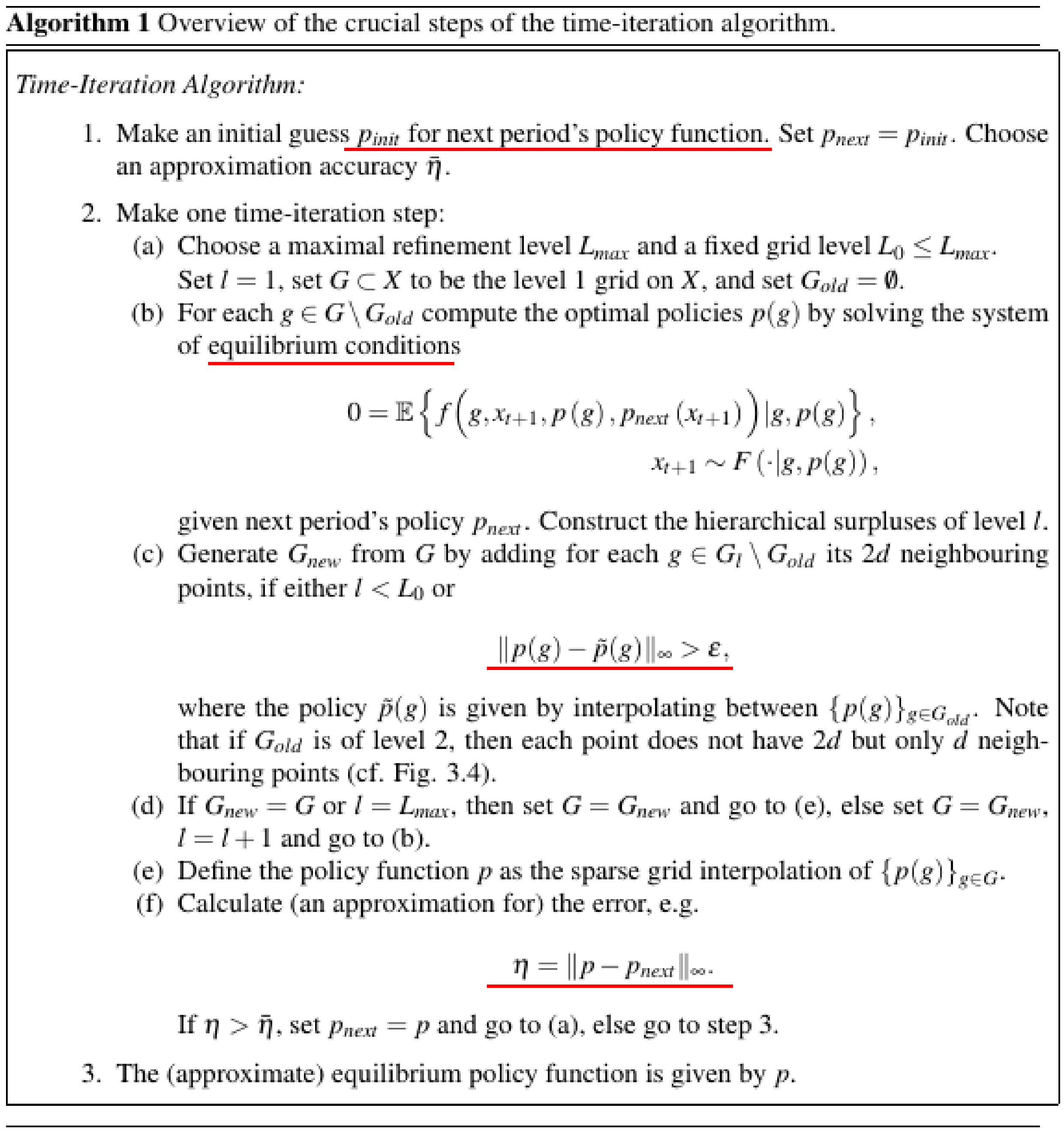
given the known policy functions from last iteration

$$(k^1_{t+2}(a_{t+1}, k_{t+1}), \ldots, k^N_{t+2}(a_{t+1}, k_{t+1}), \lambda_{t+1}(a_{t+1}, k_{t+1})) \longrightarrow$$ **Interpolant**

evaluated at next periods state

$$(a^1_t, \ldots, a^N_t, k^1_t, \ldots, k^N_t), \text{ where } a^j_{t+1} = (a^j_t)^\rho \cdot e^{\sigma(e_t + e^j_t)}.$$

Time Iteration Algorithm



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

Time-Iteration Algorithm:

1. Make an initial guess p_{init} for next period's policy function. Set $p_{next} = p_{init}$. Choose an approximation accuracy $\bar{\eta}$.
2. Make one time-iteration step:
   - (a) Choose a maximal refinement level L_{max} and a fixed grid level $L_0 \leq L_{max}$. Set $l = 1$, set $G \subset X$ to be the level 1 grid on X, and set $G_{old} = \emptyset$.
   - (b) For each $g \in G \setminus G_{old}$ compute the optimal policies $p(g)$ by solving the system of equilibrium conditions

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

     given next period's policy p_{next}. Construct the hierarchical surpluses of level l.
   - (c) Generate G_{new} from G by adding for each $g \in G_l \setminus G_{old}$ its $2d$ neighbouring points, if either $l < L_0$ or

     $$\|p(g) - \tilde{p}(g)\|_\infty > \varepsilon,$$

     where the policy $\tilde{p}(g)$ is given by interpolating between $\{p(g)\}_{g \in G_{old}}$. Note that if G_{old} is of level 2, then each point does not have $2d$ but only d neighbouring points (cf. Fig. 3.4).
   - (d) If $G_{new} = G$ or $l = L_{max}$, then set $G = G_{new}$ and go to (e), else set $G = G_{new}$, $l = l + 1$ and go to (b).
   - (e) Define the policy function p as the sparse grid interpolation of $\{p(g)\}_{g \in G}$.
   - (f) Calculate (an approximation for) the error, e.g.

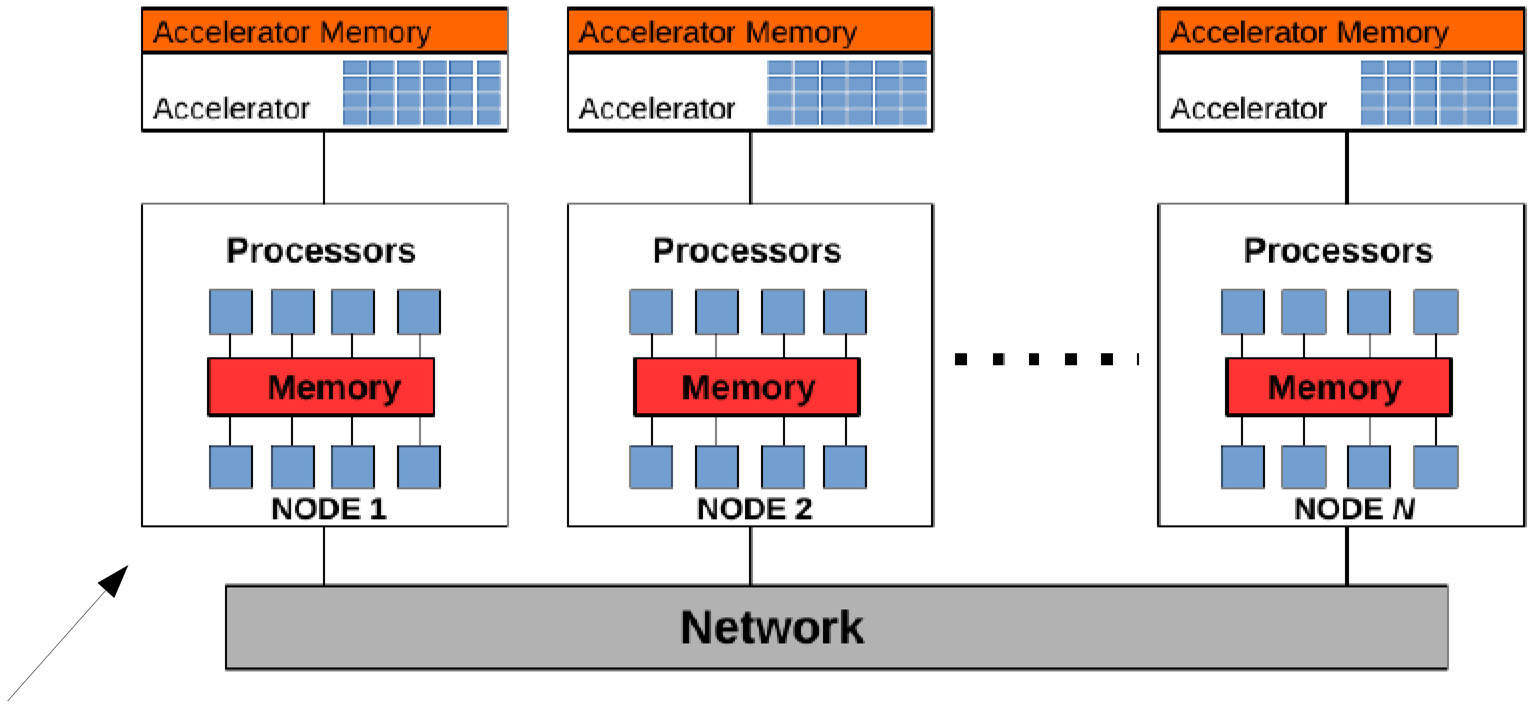
     $$\eta = \|p - p_{next}\|_\infty.$$

     If $\eta > \bar{\eta}$, set $p_{next} = p$ and go to (a), else go to step 3.
3. The (approximate) equilibrium policy function is given by p.

“To pull a bigger wagon, it is easier to add more oxen than to grow a gigantic ox” (Skjellum et al. 1999)

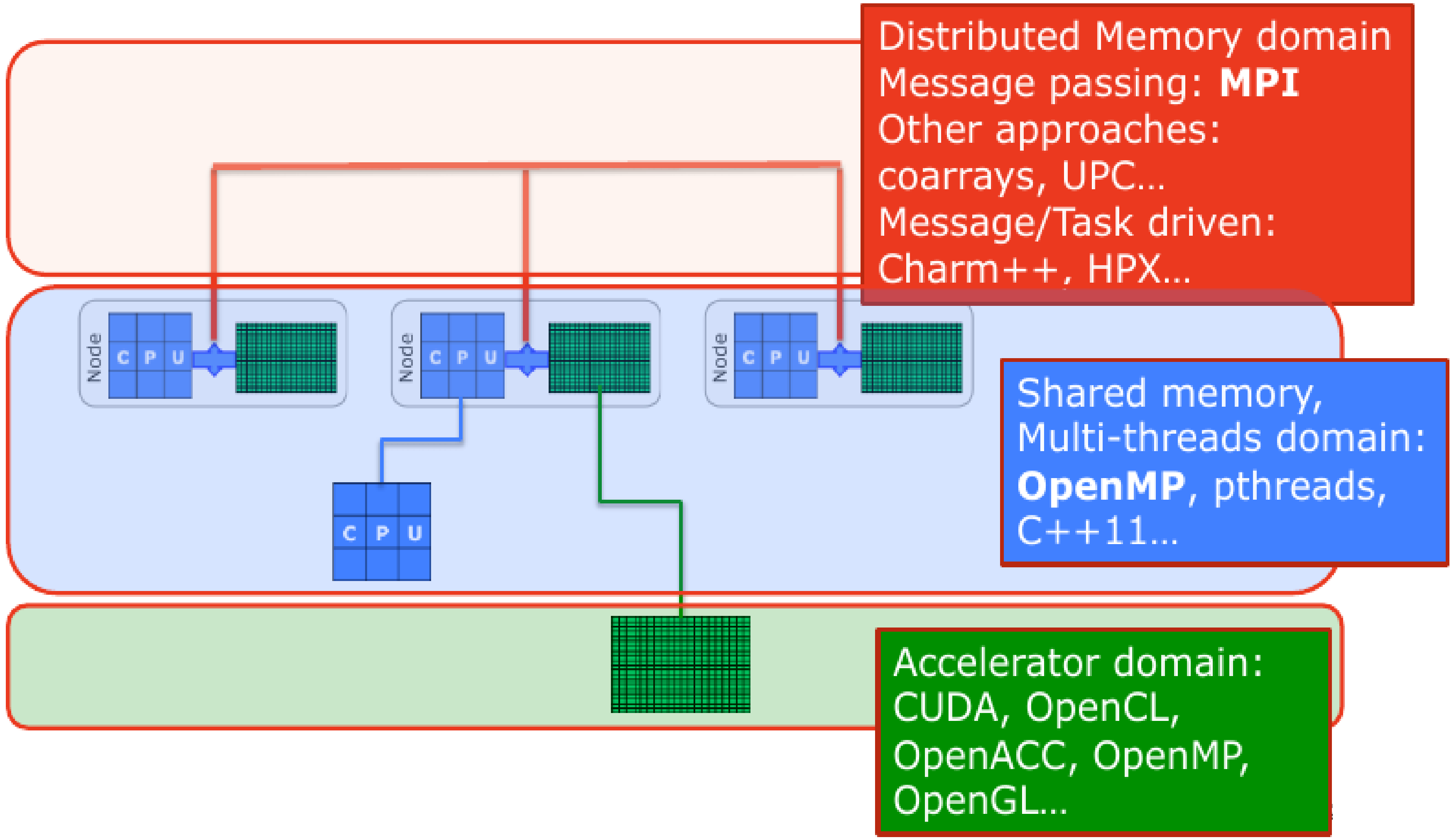
Piz Daint (Swiss Supercomputing Centre);
10x 10^{15} Flops/Sec. = 1500y of a laptop in a day.

Today's HPC systems



Your Desktop

Overall picture of programming models



Distributed Memory domain
Message passing: **MPI**
Other approaches:
coarrays, UPC...
Message/Task driven:
Charm++, HPX...

Shared memory,
Multi-threads domain:
OpenMP, pthreads,
C++11...

Accelerator domain:
CUDA, OpenCL,
OpenACC, OpenMP,
OpenGL...

→ **Creating `good' HPC software can be very difficult...**

(Slide from C. Gheller)

Speedup, Efficiency & Amdahl's Law

T(p,N) := time to solve problem of total size N on p processors.

Parallel speedup: $S(p,N) = T(1,N)/T(p,N)$

→ Compute same problem with more processors in **shorter time.**

Parallel Efficiency: $E(p,N) = S(p,N)/p$

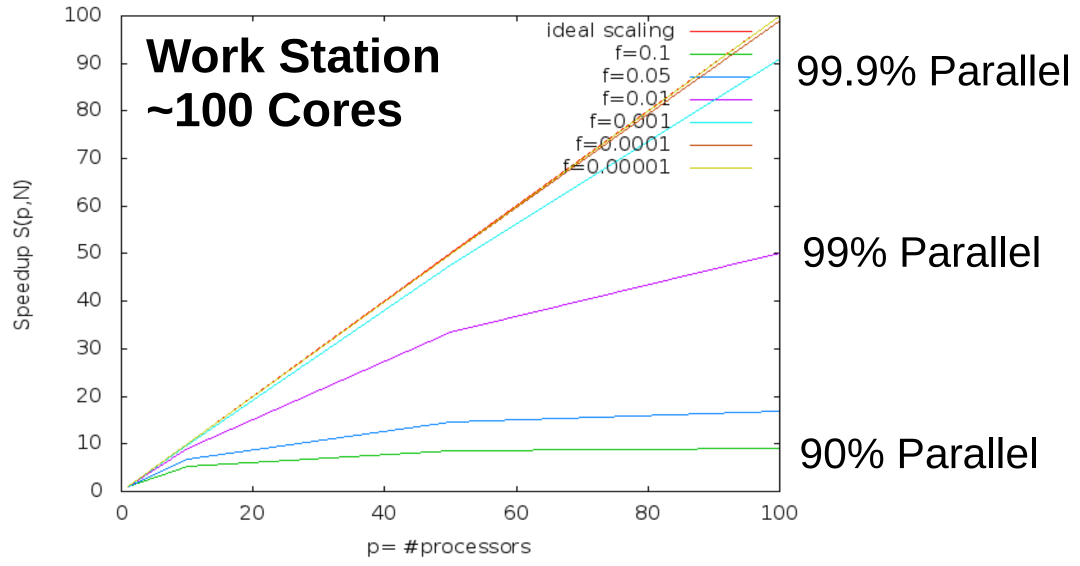
Amdahl's Law: $\boxed{T(p,N) = f * T(1,N) + \mathbf{(1-f)\ T(1,N)/p}}$

f...sequential part of the code that can not be done in parallel

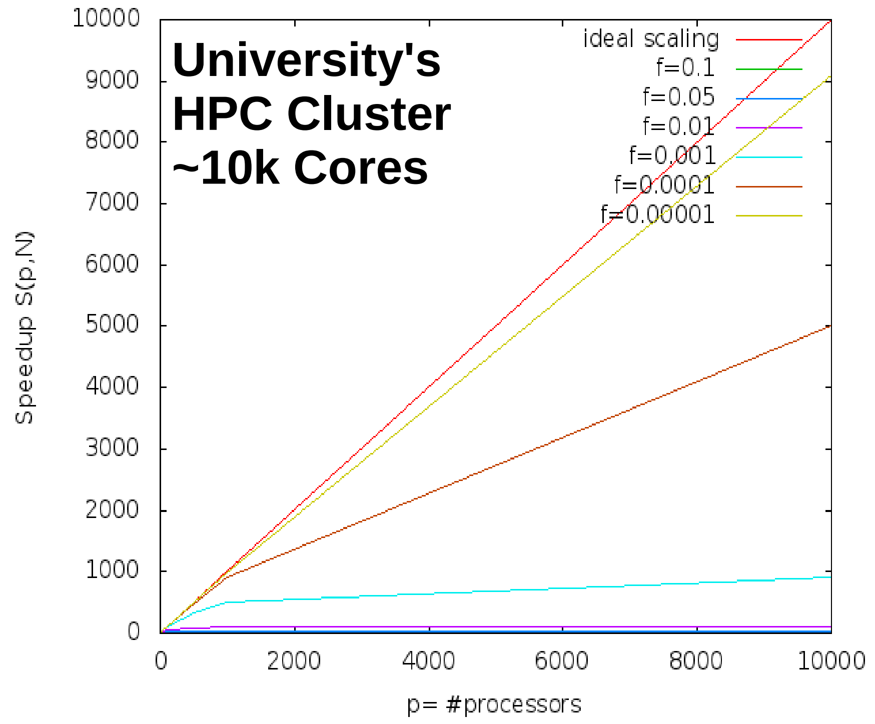
$$\boxed{S(p,N) = T(1,N)/T(p,N) = 1 / (f + (1-f)/p)}$$

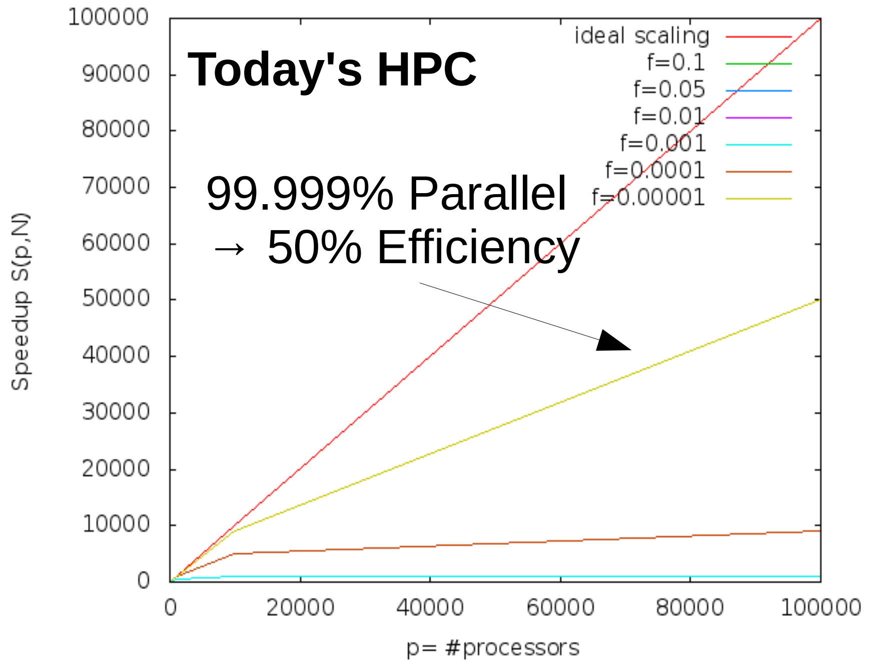
For $p \rightarrow$ infinity, speedup is limited by $S(p,N) < 1/f$

Amdahl's Law: Scaling is tough



For p → infinity:
Speedup is limited by S(p,N) < 1/f





Parallel time iteration/DP algorithm

-Our implementation:
Hybrid parallel
(MPI & Intel TBB & GPU (CUDA/THRUST)).

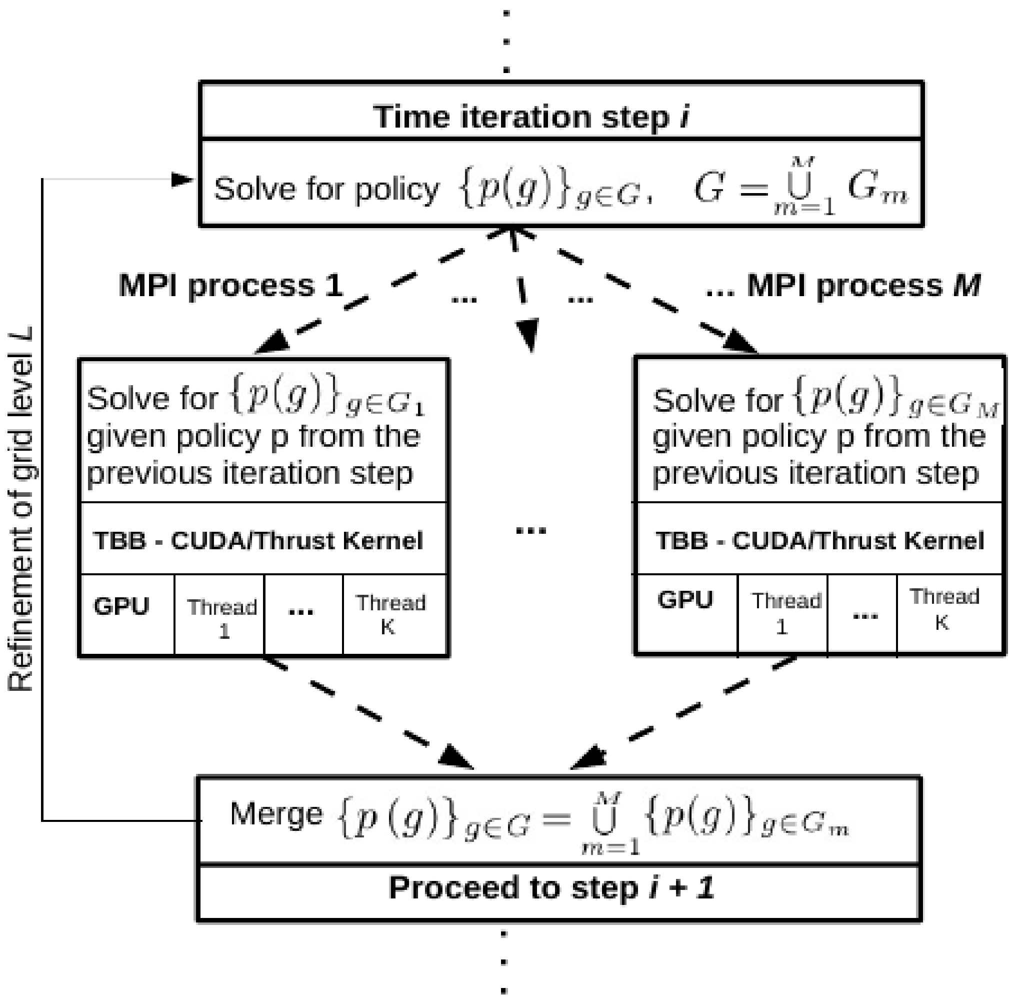
-newly generated points are distributed via MPI

Solve optimizations/ nonlinear equations locally
(e.g. IPOPT (Waechter & Biegler (2006)).

In parallel: `messy' !

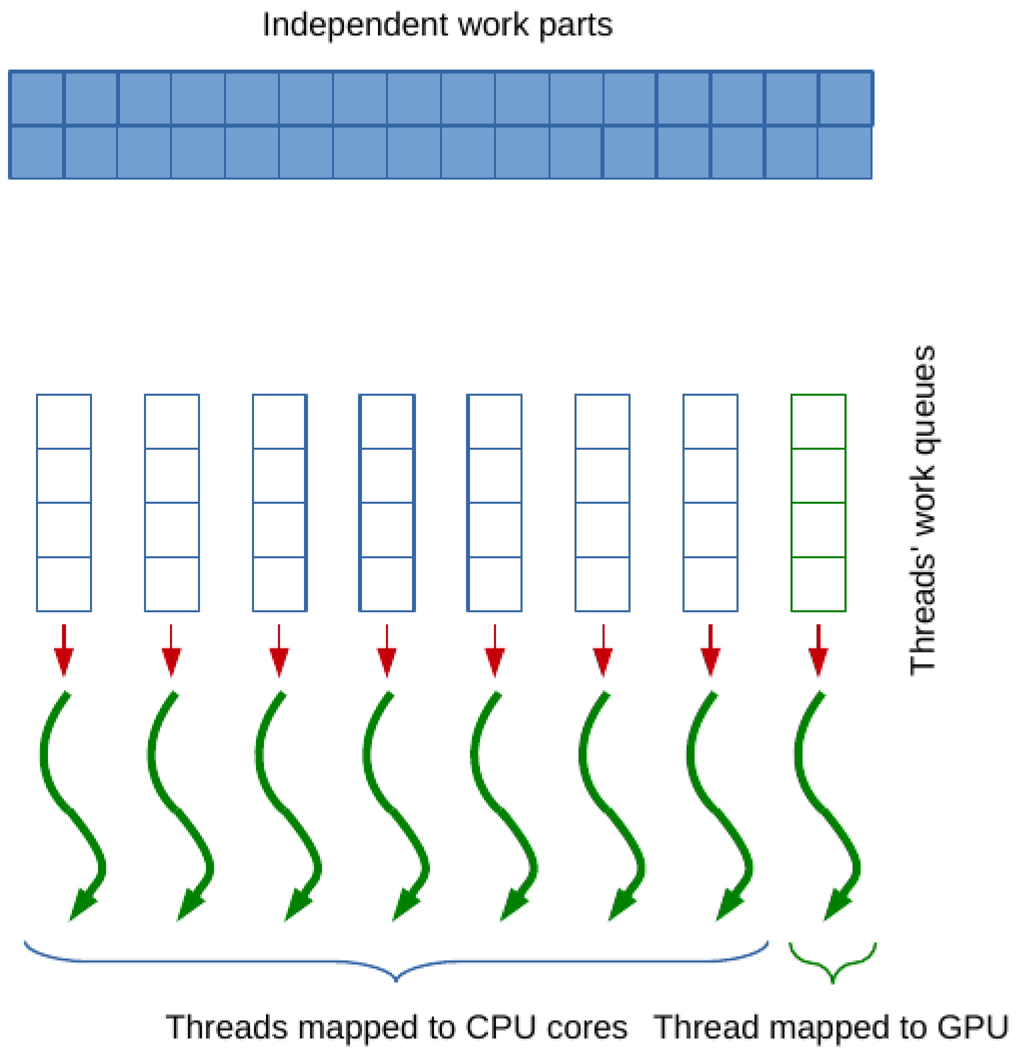
→ policy from previous iteration visible on all MPI processes.
→ we have to ensure some sort of `load balancing'.

One single time-step


Time iteration step i
Solve for policy $\{p(g)\}_{g\in G}$, $G = \bigcup_{m=1}^{M} G_m$
MPI process 1
...
... MPI process M
Solve for $\{p(g)\}_{g\in G_1}$ given policy p from the previous iteration step
TBB - CUDA/Thrust Kernel
GPU
Thread 1
...
Thread K
Solve for $\{p(g)\}_{g\in G_M}$ given policy p from the previous iteration step
Merge $\{p(g)\}_{g\in G} = \bigcup_{m=1}^{M} \{p(g)\}_{g\in G_m}$
Proceed to step $i + 1$
Refinement of grid level L

Intel® Threading Building Blocks (TBB)

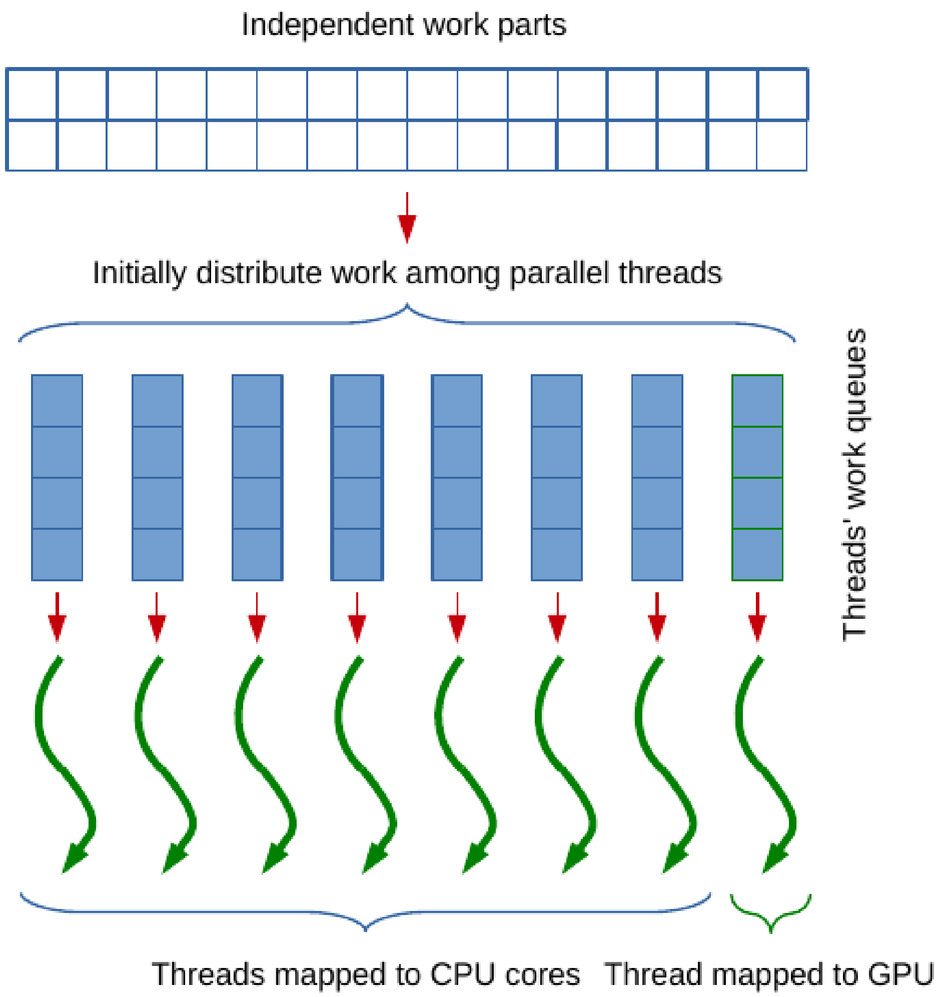
-TBB maps different threads, similar to OpenMP.



Intel® Threading Building Blocks (TBB)

-TBB maps different threads, similar to OpenMP.

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


Independent work parts
Initially distribute work among parallel threads
Threads' work queues
Threads mapped to CPU cores
Thread mapped to GPU

Intel® Threading Building Blocks (TBB)

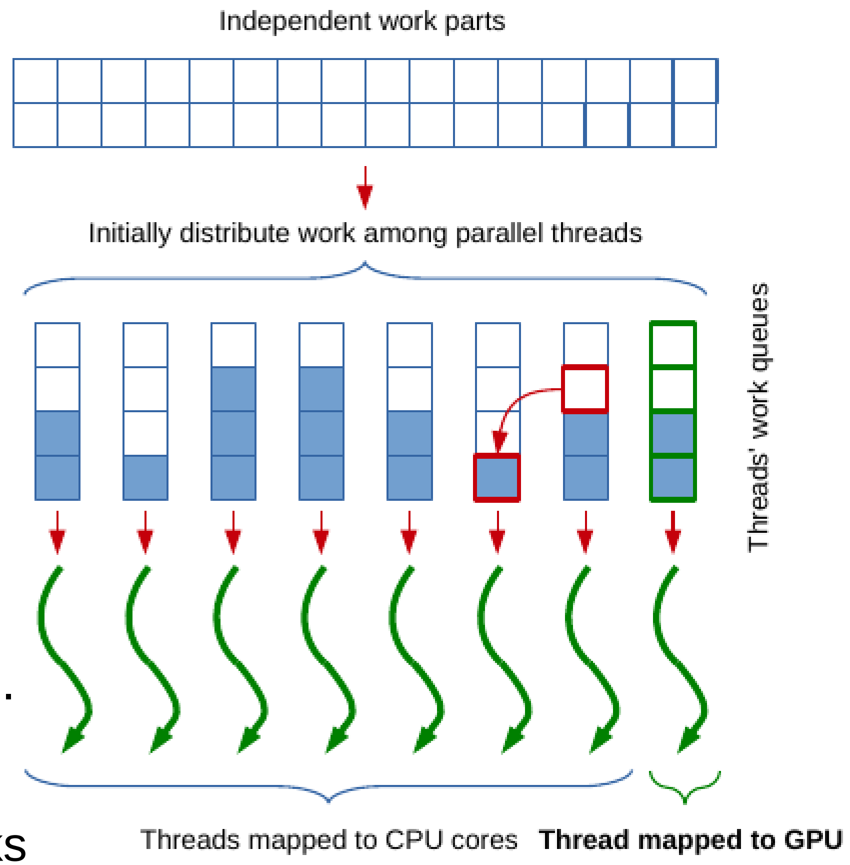
-TBB maps different threads, similar
 to OpenMP.

-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

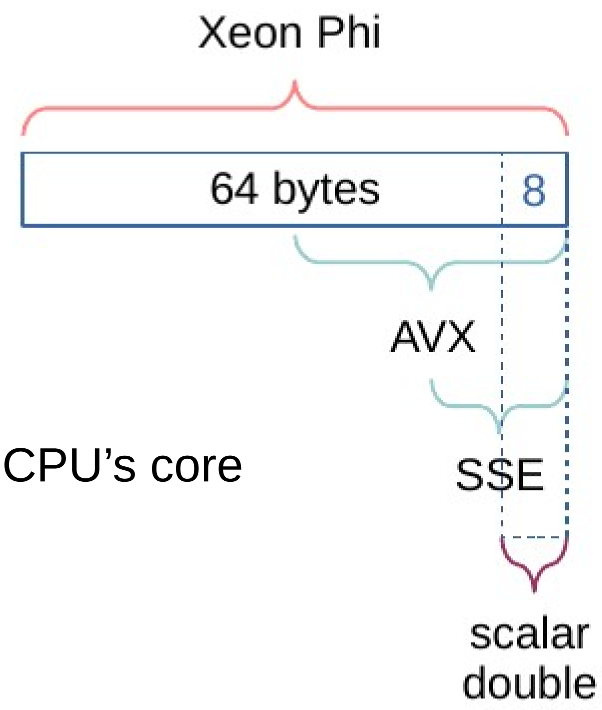
-TBB approach to work balancing: once one
 thread runs out of tasks, **“steal” a task from
 another thread, which makes slower progress**.

-We map one extra thread onto **GPU**
→ **CPU** cores and **GPU** process interpolation tasks
together.


Independent work parts
Initially distribute work among parallel threads
Threads' work queues
Threads mapped to CPU cores
Thread mapped to GPU

AVI: AVX vectorization

Brumm, Mikushin, Scheidegger, Schenk (2014 – submitted)



-All modern CPUs have vector registers to process from 16 (SSE – old CPUs) to 64 (Xeon Phi) bytes of data simultaneously.

- **2012 and newer CPU can process 4 doubles with one instruction** (AVX - Advanced Vector Extensions)

- Running code without vectorization use of only 1/4 of CPU's core compute power (e.g. Piz Daint's CPUs)

- Compilers try to vectorize scalar code automatically, but usually succeed in simple cases only.

→ we need to develop vector-aware code, in order to make it efficient!

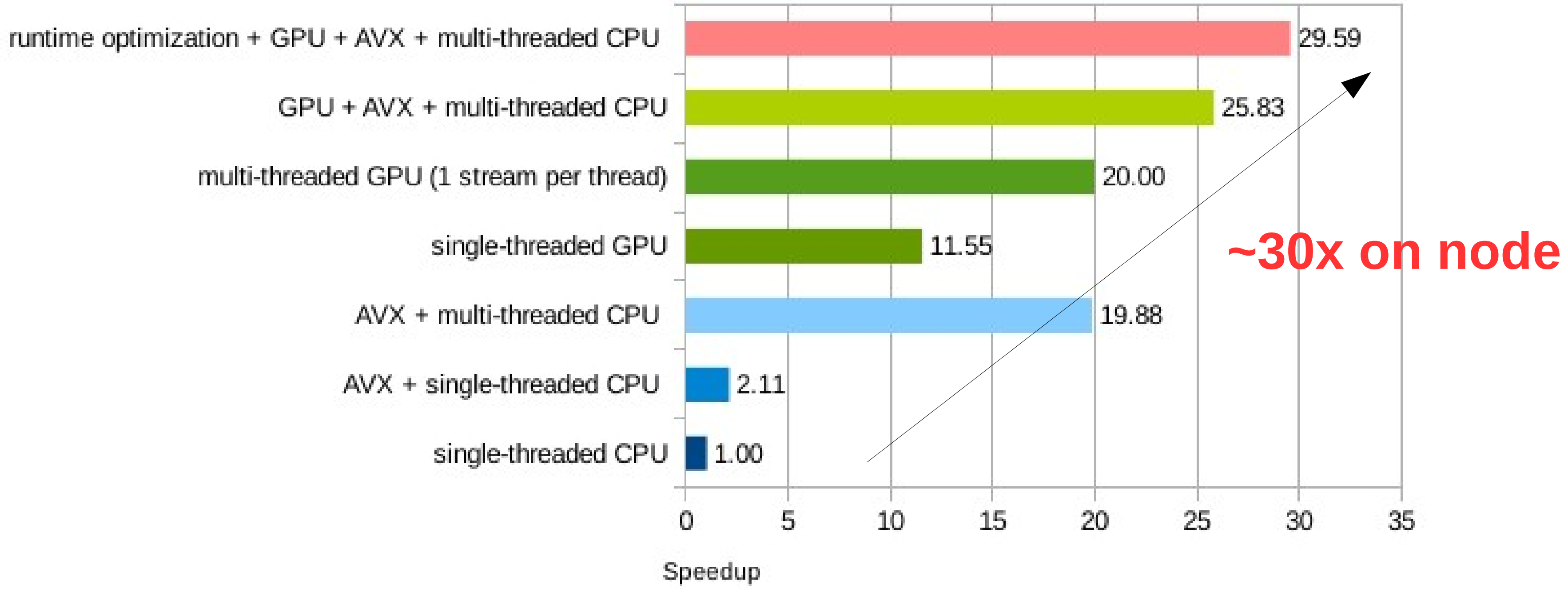
Example of scalar loop and its vectorized equivalent:

```
for (int i = 0; i < N; i++) y[i] += alpha * x[i];
```

```
for (int i = 0; i < N / 8; i++)
    _mm512_store_pd((void *)(y[8*i]), _mm512_add_pd( _mm512_load_pd((void *)(y[8*i])),
        _mm512_mul_pd(_mm512_set1_pd(alpha), _mm512_load_pd((void *)(x[8*i]))) ));
```

Single-node Code Optimization

Brumm, Mikushin, Scheidegger, Schenk (2015)



Notes:
- Level 2 (of an IRBC with binding constraints - 128 grid points)
- Fully optimized code on 1 core (bitwise shifts, code inline, replace divisions where possible,...)
- **1 node today consists of several cores plus often a GPU (Piz Daint: Kepler K20X)**.
- AVX: Vectorization for 2012+ processors (“advanced vector extension”)

A Strong Scaling Example

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

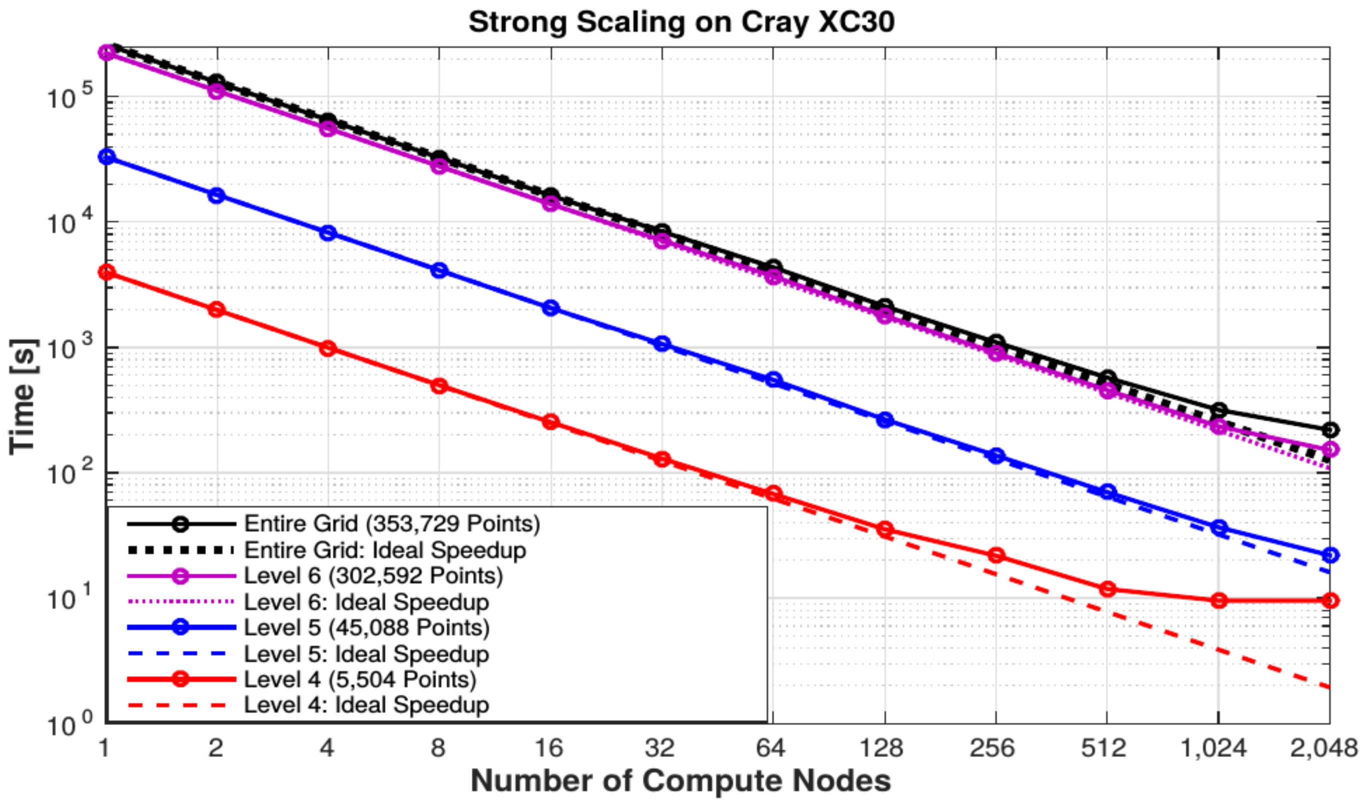
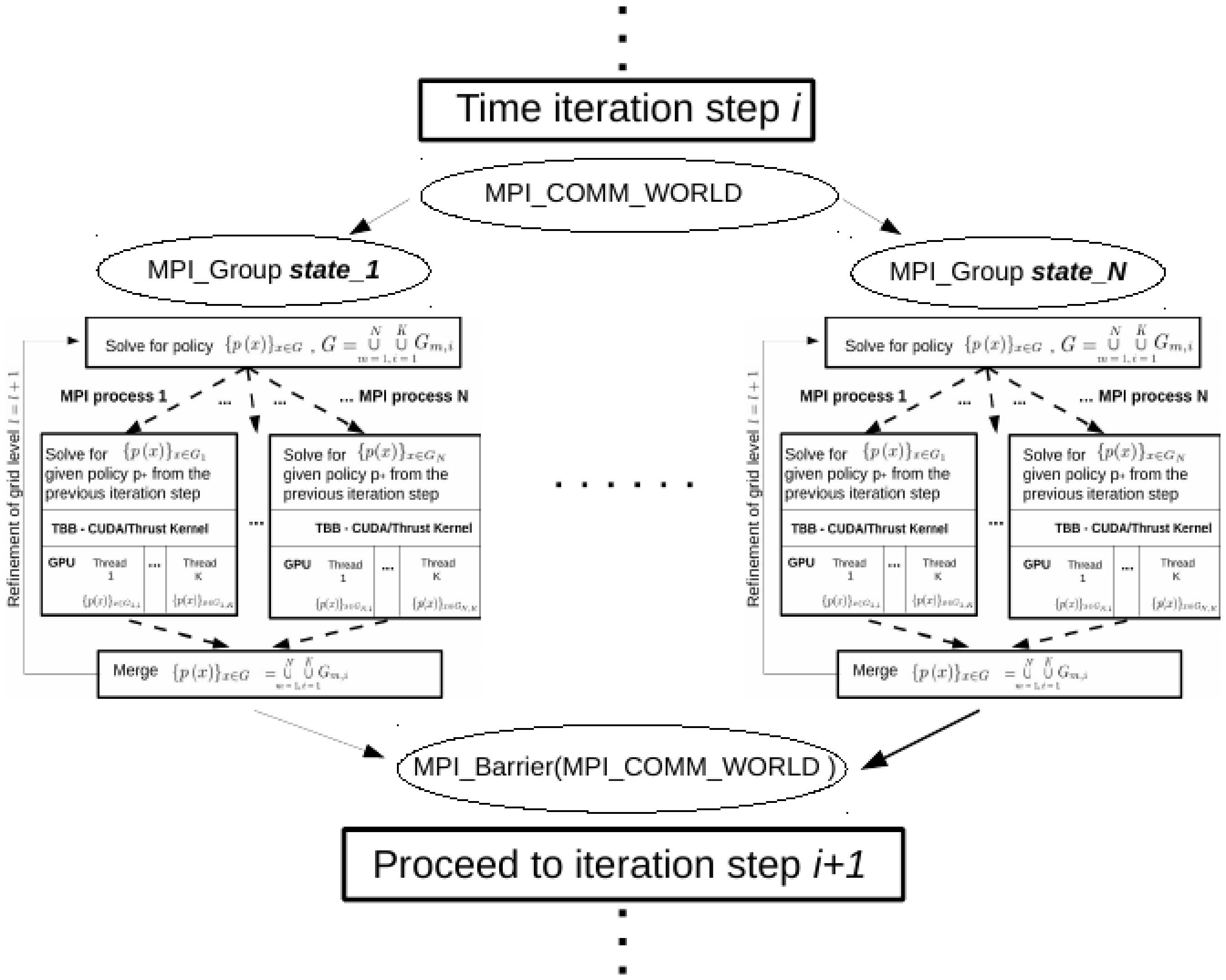


Fig.: strong scaling of the code.
Problem: one timestep of an 16d IRBC, fixed refinement level 5.
Overall speedup more than **3 orders of magnitude** (cf. 1 single node) .

Going beyond IRBC → OLG

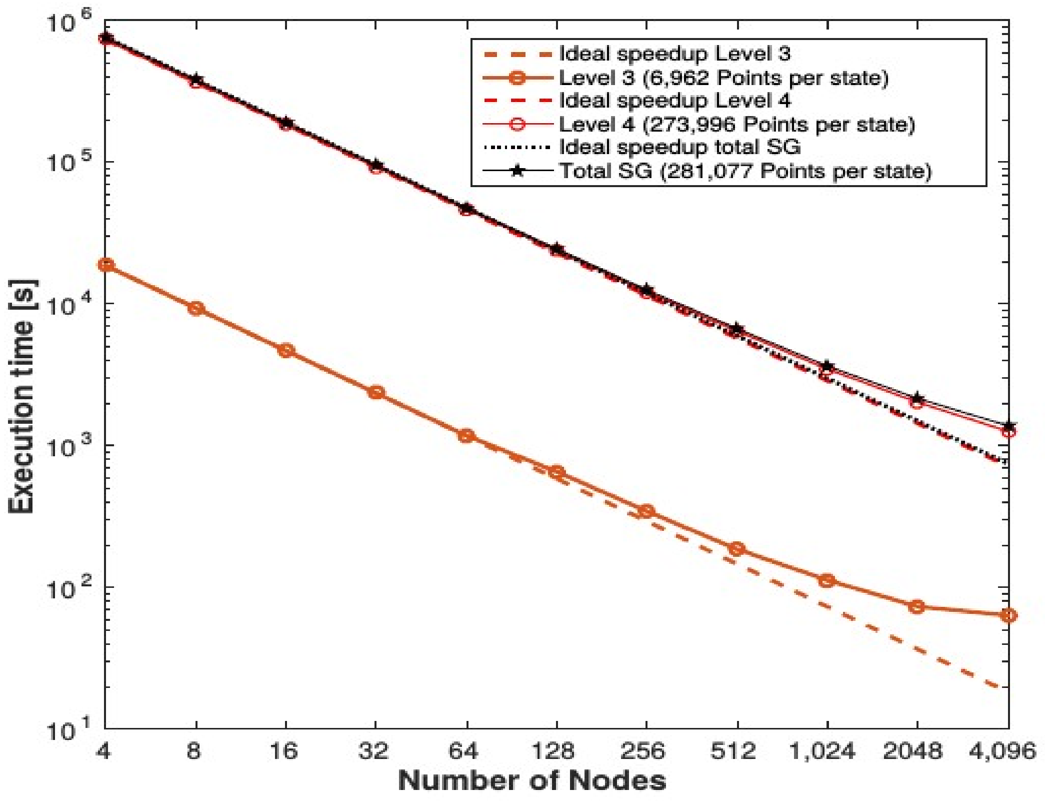
Brumm, Kübler, Scheidegger (2015)

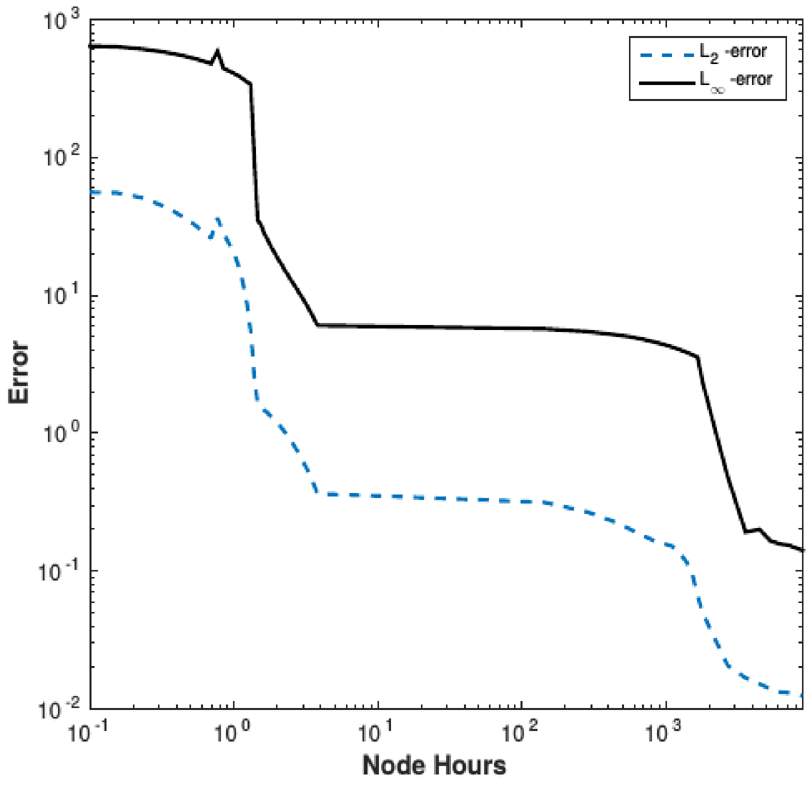
→ OLG Model with 60 continuous state variables.
→ 4 discrete states.


Time iteration step *i*
MPI_COMM_WORLD
MPI_Group ***state_1***
MPI_Group ***state_N***
Solve for policy $\{p(x)\}_{x\in G}$, $G = \bigcup_{m=1}^{N}\bigcup_{i=1}^{K} G_{m,i}$
MPI process 1
... MPI process N
Solve for $\{p(x)\}_{x\in G_1}$ given policy p+ from the previous iteration step
Solve for $\{p(x)\}_{x\in G_N}$ given policy p+ from the previous iteration step
TBB - CUDA/Thrust Kernel
GPU
Thread 1
Thread K
Refinement of grid level $l = l + 1$
Merge $\{p(x)\}_{x\in G} = \bigcup_{m=1}^{N}\bigcup_{i=1}^{K} G_{m,i}$
MPI_Barrier(MPI_COMM_WORLD)
Proceed to iteration step *i+1*

Discrete states: scaling & convergence

Scheidegger et. al (2017) – in preparation (draft available at https://sites.google.com/site/simonscheidegger/publications)

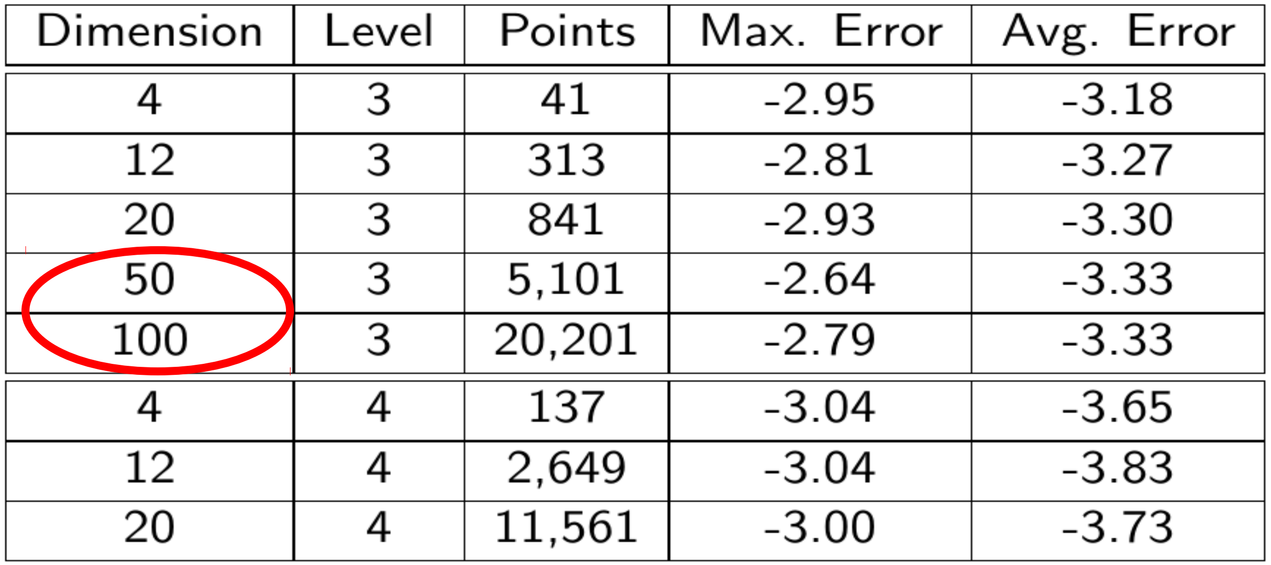




→ scales ~linearly with number of discrete states, since they are fully independent.
→ Can use 80% of system with ~60% efficiency (this model).
→ Prepare code for exa-scale systems (Xeon Phi Knight landing).

Results for Smooth IRBC Model

Non-adaptive sparse grid of fixed level produces stable accuracy when dimension is increased massively:



Dimension	Level	Points	Max. Error	Avg. Error
4	3	41	-2.95	-3.18
12	3	313	-2.81	-3.27
20	3	841	-2.93	-3.30
50	3	5,101	-2.64	-3.33
100	3	20,201	-2.79	-3.33
4	4	137	-3.04	-3.65
12	4	2,649	-3.04	-3.83
20	4	11,561	-3.00	-3.73

All errors are given in log 10 -scale.

Results for Non-Smooth IRBC Model: Finer Grids

Non-adaptive sparse grid has problems with non-smooth model:

Dimension	Level	Points	Max. Error	Avg. Error
4	5	401	-2.11	-2.93
4	7	2,929	-2.32	-3.12
4	9	18,945	-2.45	-3.32

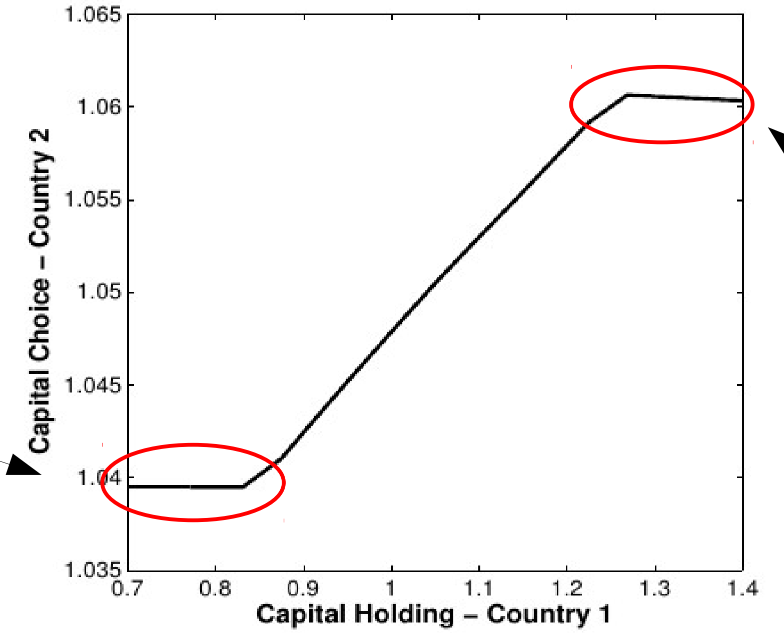
Adaptive sparse grid can overcome problems with non-smooth model
→ they provide higher accuracy with less points:

ϵ	Max. Level Reached	Points	Max. Error	Avg. Error
0.01	7 (2,929)	245	-2.23	-2.88
0.005	9 (1,945)	559	-2.42	-2.98
0.0025	13 (643,073)	2,346	-2.68	-3.32
0.001	14 (3,502,081)	14,226	-2.91	-3.73

All errors are given in log 10 -scale.

IRBC with irreversible investment

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

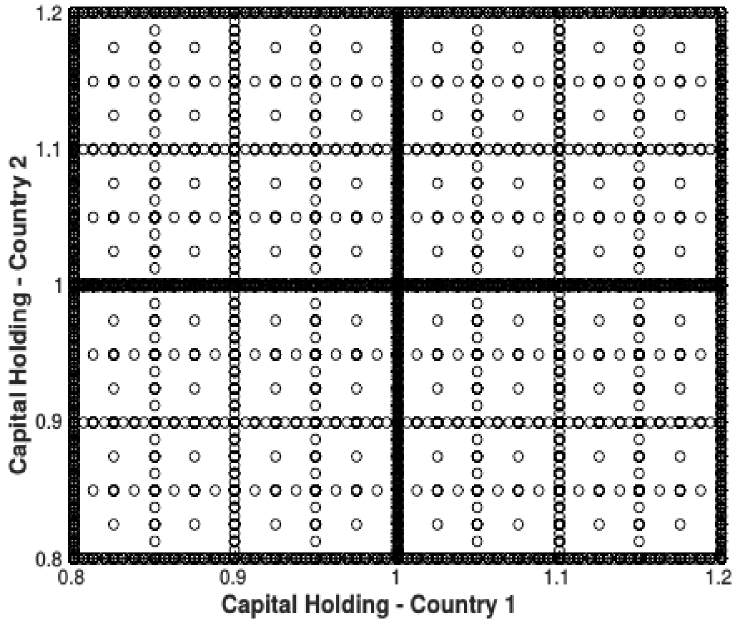


Capital in country 1 high:
→ irrev. constraint binding for country 1.
→ this limits transfer of resources to country 2.

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.**

IRBC with binding constraints



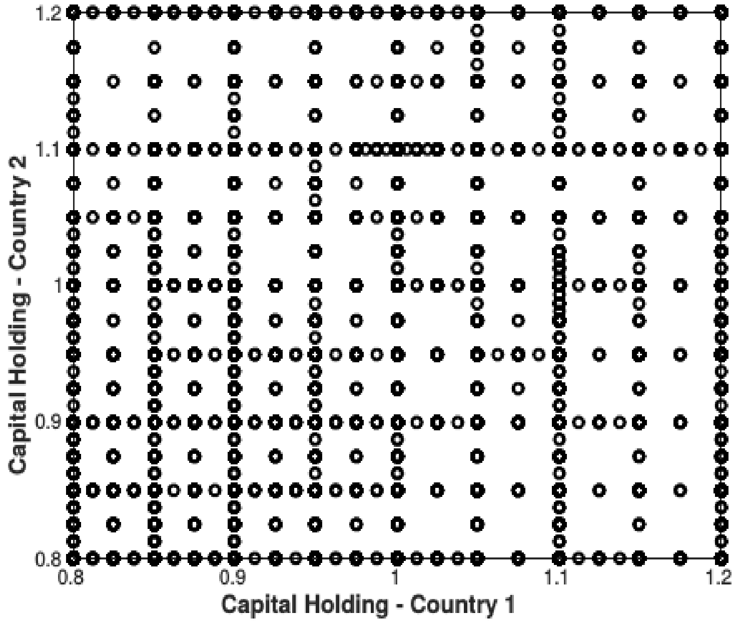


Fig.: 2-d projections of two different grids.
Left: \`classical' sparse grid of level 9 (18,945 points),
Right: adaptive grid with refinement threshold ε = 0.001 (14,226 points).
The x-axis shows capital holding of country 1, the y-axis shows capital holding of country 2, while the productivities of the two countries are kept fixed at their unconditional means.

Models with binding constrains: massive speedup due to adaptivity

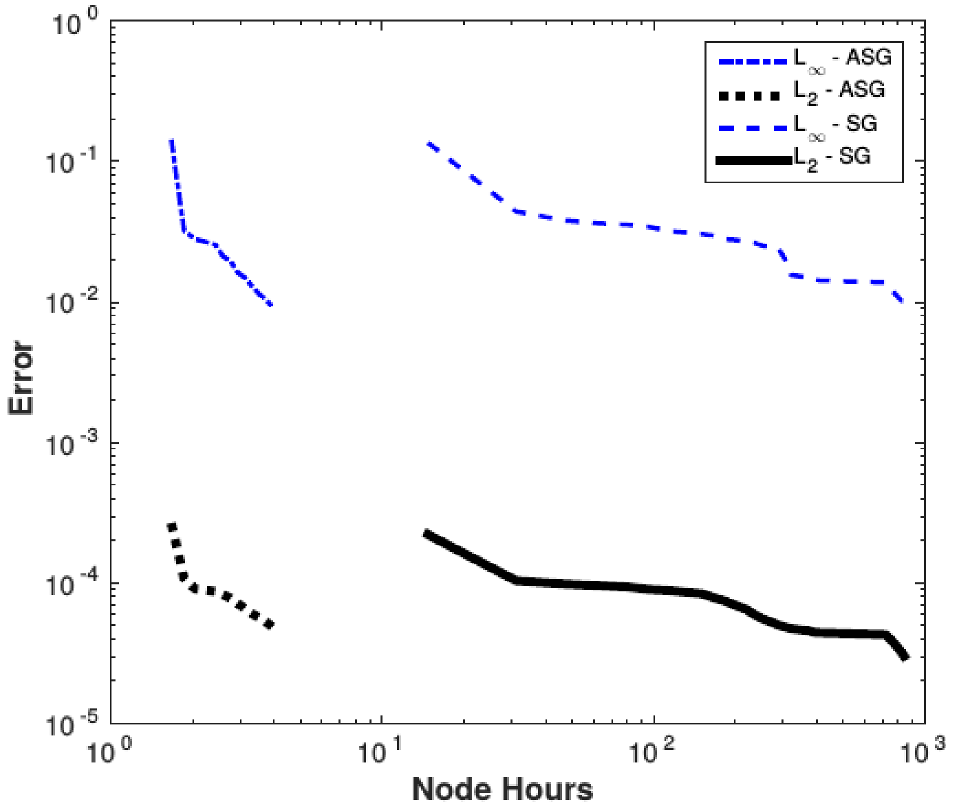


Fig.: 8d model with binding constraints.
model run with/without adaptive sparse grids.
Relative error among two consecutive time-steps.
10k points drawn from uniform distribution.

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

- **Method perfectly suited to solve high dimensional dynamic models with large amount of heterogeneity! (Method: Scalable & Flexible).**

Method very helpful to address:

- High-dimensional, heterogeneous time-iteration & dynamic programming problems.
- 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.**
- Option pricing.