# Parallel Adaptively Refined Sparse Grids

Gerhard Zumbusch[1]

University Bonn, Dept. Applied Mathematics, Wegelerstr. 6, D-53115, Germany

**Abstract.** A parallel version of a finite difference discretization of PDEs on sparse grids is proposed. Sparse grids or hyperbolic crosspoints can be used for the efficient representation of solutions of a boundary value problem, especially in high dimensions, because the number of grid points depends only weakly on the dimension. So far only the 'combination' technique for regular sparse grids was available on parallel computers. However, the new approach allows for arbitrary, adaptively refined sparse grids. The efficient parallelisation is based on a dynamic load-balancing approach with space-filling curves.

## 1 Introduction

Quite lot of phenomena in science and engineering can be modelled by boundary value problems of partial differential equation type. Often this gives rise to a PDE in one or two dimensions ($d = 1, 2$), which can be treated numerically more easily. However, the accurate solution of problems in three and higher dimensions would be extremely expensive or simply impossible to solve numerically, referred to as the 'curse of dimension'. We mention higher-dimensional problems in financial engineering, in quantum physics, in statistical physics and even the four-dimensional problems in general relativity.

Sparse grids are a multi-dimensional approximation scheme, which is known under several names such as 'hyperbolic crosspoints', 'splitting extrapolation' or as a boolean sum of grids. Probably Smolyak [16] was the historically first reference. Directly related to the boolean construction of the grids was the construction of a multi-dimensional quadrature formula. Both quadrature formulae and the approximation properties of such tensor product spaces were subject to further research, see Temlyakov [17] and others. The curse of dimension was also subject to general research on the theoretical complexity of higher-dimensional problems. For such theoretical reasons, sparse grids play an important role for higher-dimensional problems. Besides the application to quadrature problems, sparse grids are now also used for the solution of PDEs. They were introduced for the solution of elliptic partial differential equations by Zenger [18], where a Galerkin method, adaptive sparse grids and tree data structures were discussed. At the same time a different discretization scheme based on the extrapolation of solutions on several related, regular grids was proposed, the 'combination' technique see [6].

In this paper we consider a finite difference (FD) scheme [5,14]. It is simpler to implement and to apply to different types of equations, but there is not that much known analytically yet. We focus on the parallelisation of such a FD scheme on sparse grids. So far only the 'combination' technique can be run on a distributed-memory parallel computer, see [4], which is essential for many large scale simulations. The advantage of the FD scheme is the flexibility of the grid. The sparse grid may be refined adaptively, while the 'combination' is relies on regular sparse grids. However, the adaptive grid refinement poses a severe load-balancing problem, which has to be resolved during runtime of the simulation. Using a space-filling curve distribution of the grid points similar to [11,12,8,13] for standard grids and some parallel decompositions of the hierarchical sparse grid algorithms, we are now able to run adaptive sparse grid computations on a parallel computer.

## 2 Sparse Grids

The multi-dimensional approximation scheme of sparse grids can be constructed as a subspace of the tensor-products of one-dimensional spaces represented by a hierarchical multi-resolution scheme, such as the hierarchical basis see the historical reference [3], or generally any basis system of pre-wavelets or wavelet [9]. Each one-dimensional basis function can be derived from a model function $\phi$ by a scaling of $2^{-l}$ (also called level $l$) and a translation by a multiple of $2^{-l}$. In the case of the hierarchical basis, the model function $\phi$ is a hat function. We denote the one-dimensional space of functions up to level $l$ by $T_l$. On level $l$, the standard grid space can be written as

$$\hat{T}_l = \langle T_i \otimes T_j \otimes \ldots \rangle_{i,j,\ldots \leq l} \,. \tag{1}$$

In contrast to the regular grid, the corresponding sparse grid space consists of fewer functions. On level $l$, it can be written as

$$\tilde{T}_l = \langle T_i \otimes T_j \otimes \ldots \rangle_{i+j+\ldots \leq l} \,. \tag{2}$$

This is a subset of the regular grid space. A regular grid has about $2^{d \cdot l}$ nodes, which is substantially more than the $2^l \cdot l^{d-1}$ nodes of the sparse grid.

The major advantage of sparse grids compared to regular grids is their smaller number of nodes (or grid points) for the same level $l$ and resolution $2^{-l}$. This is especially true in higher dimensions $d \gg 1$.

Of course, the question whether sparse grids have an advantage compared to regular grids does also depend on the discretization accuracy of a solution obtained on a grid. Furthermore the number-of-operations complexity is of interest, because it is an estimate for the computing time a specific algorithm needs. For details see [5,14].

## 3 Finite Differences Operators

We define the hierarchical transformation **H** as the hierarchical basis transformation on the regular grid from nodal values to hierarchical values, which are restricted to the sparse grid nodes. All wavelet-type of basis functions provide such fast $\mathcal{O}(n)$ transformation to and from the nodal basis representation. The transformation is especially simple for the one-dimensional hierarchical basis: Given the nodal values $u_j$ with $j = 0, 1, \ldots, 2^{l+1}$, the hierarchical representation for interior points can be obtained by

$$\hat{u}_j \;=\; u_j - \frac{1}{2}\big(u_{\text{left father}} + u_{\text{right father}}\big) \tag{3}$$

and the boundary nodes $u_0$ and $u_{2^{l+1}}$ remain unchanged. The nodal values are replace by their hierarchical excess or deterioration, compared to the value obtained by interpolation between on the next coarser level grid. The inverse transformation can be implemented similarly. However, the coarse nodes have to be computed before the finer grid nodes. Furthermore, the transformation can be implemented in place, without an auxiliary vector. The hierarchical basis transformation $\mathbf{H}$ is also abbreviated by the stencil [1/2 1 1/2].

Based on the hierarchical basis transformation $\mathbf{H}$, we define the action of a one-dimensional finite difference operator for the discretization of a differential operator: We apply the associated standard difference stencil $\mathbf{D}_i$ along the $x_i$-axis to values located on the sparse grid nodes in a specific basis representation. To this end the values are given in nodal basis in direction $i$ and in hierarchical basis representation in all other directions $I \setminus \{i\}$. The associated transformation is denoted by $\mathbf{H}_{I\setminus\{i\}}$. The stencil $\mathbf{D}_i$ for each node itself is chosen as the narrowest finite difference stencil available on the sparse grid. It is equivalent to the corresponding stencil on a regular, anisotropic refined grid. The finite difference stencil can by a 3-point Laplacian $\frac{1}{h^2}[1 \;\; -2 \;\; 1]$, an upwind-stabilised convection term $\frac{\partial}{\partial x_i}$, some variable coefficient operators and so on. In nodal values the finite difference operator reads

$$\frac{\partial^2}{\partial x_i^2} u \;\approx\; \mathbf{H}^{-1}_{I\setminus\{i\}} \circ \mathbf{D}_{ii} \circ \mathbf{H}_{I\setminus\{i\}}\, u\,. \tag{4}$$

A general difference operator is then obtained by dimensional splitting. A Poisson equation, as a simple example, can be discretized in nodal basis representation as usual as a sum of operators 4. Here the one-dimensional difference operators $\mathbf{D}_i$ may be chosen as a three point centred Laplacian $a \cdot \frac{1}{(x_{i+1} - x_{i-1})^2}[1 \;\; -2 \;\; 1]$. On adaptively refined grids, the nearest neighbour nodes are chosen, which may lead to asymmetric stencils, i.e. non-uniform one-dimensional stencils. Further higher order modifications of the stencils have been tested, too. In the presence of a transport term in the equation, the unsymmetry is believed to be no problem. There are many ways to create discretizations of all kind of equations, e.g. for the Navier-Stokes equations [14] or some hyperbolic conservation laws [7]. Known facts on consistency and stability of this scheme are summarised in [14].

## 4 Parallelisation

The parallelisation of an adaptive code usually is non-trivial and requires a substantial amount of code for the parallelisation only. In this respect the 'combination' technique based on regular grids is much easier. If we are interested in a parallel version for adaptive sparse grids, however, we have to consider a more complicated approach to be described now. Hierarchies of refined grids, where neighbour nodes may reside on different processors, have to be managed. That is, appropriate ghost nodes have to be created and updated, when the parallel algorithm performs a communication operation. This happens both in the numerical part, where an equation system is set up and solved, and in the non-numerical part, where grids are refined and partitioned, see for standard grids also [1,10,8].

The key point of any dynamic data partition method is efficiency. We look for a cheap, linear time heuristic for the solution of the partition problem. Furthermore the heuristic should parallelise well. Here, parallel graph coarsening is popular. It results in a coarser graph on which then a more expensive heuristic on a single processor can be employed. However, graph coarsening is at least a linear time algorithm itself and lowers the quality of the heuristic further. This is why we look for even cheaper partition methods. They are provided by the concept of *space-filling curves*, see Figure 1.

In addition, key-based addressing is used, which substitutes the memory address stored in a pointer (see [18,2]) with an integer value uniquely describing the entity. In our case, each node can be characterised by its position in space, that is the local coordinates. The advantage in the sequential version is simplicity and little administration overhead. The parallel version is based on the distribution of nodes to the processors. Each processor owns a subset of the sparse grid. Each node is present on exactly one processor. Furthermore, the space-filling curve, which provides a unique mapping of nodes to processors, immediately reveals, which processor to ask for a node. Other grid partitioning heuristics in contrast would require a substantial bookkeeping effort to decide where a node belongs to.

The FD operator 4 is composed of three basic operations, the transform to hierarchical basis $\mathbf{H}_j$, the one-dimensional finite difference stencil $\mathbf{D}_i$ and the transform back to nodal basis $\mathbf{H}_j^{-1}$, which have to be implemented in parallel versions separately.

- Transform to hierarchical basis $\mathbf{H}_j$: Each processor computes the values related to its own nodes. Prior to the computation, in a communication step the required ghost nodes are filled. The ghost nodes for this operation are determined by the direct parent nodes of nodes on the processor.
- Transform to nodal basis $\mathbf{H}_j^{-1}$: This operation can be done in place and requires more communication than the previous one. The sequential implementation cycles through a tree top down, so that the parent nodes are processed before their children. A straightforward parallelisation would

be to insert a communication step before each tree level is traversed. However, this results in a number of communication steps (= communication latencies) proportional to the maximum number of levels, which is unacceptable for large sparse grids.
We propose an alternative implementation here, which is based on a single communication step before the computation: Along with with the parents of a node, the whole tree of their grand-parents and so on are required as ghost nodes on a processor. When the ghost nodes are filled, the computation can be done top down, such that the values on all nodes owned by the processor and additionally their parents, grand-parents and so on are computed. Hence, this implementation requires a larger amount of computation and a larger volume of communication than the straightforward version. However, the overall execution time is smaller because of the number of communication steps is reduced to one.
- Finite difference operator $\mathbf{D}_i$: First the appropriate ghost nodes for the difference stencil is filled and afterwards the stencils are applied to all nodes, which belong to the processor. The main point here is the searching procedure for the neighbour nodes that are necessary for adaptive refined sparse grids. We create the necessary ghost nodes, so that the sequential search algorithm can be re-used in this situation.

Following the a posteriori error estimation together with some refinement rules, new nodes are created. This can be done also in parallel. Afterwards, a repartitioning has to takes place. The execution time of this repartitioning step usually is so low that it is below .01 of the execution time spent in the numerical algorithms, see also [19].

## 5 Numerical Experiments

All numbers reported are scaled CPU times measured on our parallel computing cluster 'Parnass$_2$'. It consists of dual processors Pentium II 400MHz, interconnected by a Myrinet network in a fat-tree configuration with a Linpack performance of 29.7 GFlop/s. The MPI message passing protocol implementation Mpich-PM showed a bandwidth between each two boards of 850 Mbit/s, see also [15].

We consider adaptively refined sparse grids for a problem with singularities, where the sparse grids are refined towards a singularity located in the lower left corner. Table 1 depicts wall clock times in the adaptive case. Due to the solution-dependent adaptive refinement criterion, the single processor version contained slightly more nodes, indicated by *. For the same reason, the equation systems have been solved up to rounding error instead of the weaker discretization error condition in the uniform sparse grid experiment.

We obtain a good scaling, both in the number of unknowns and in the number of processors, i.e. the times are proportional to the number of unknowns for a fixed number of processors and are indirect proportional to

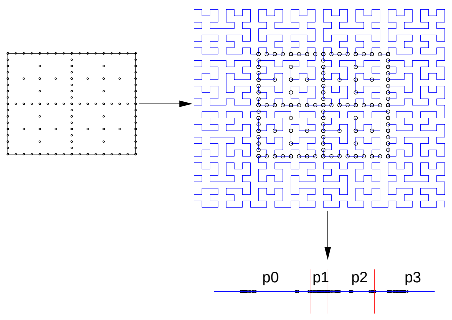

**Fig. 1.** An example of a sparse grid partitioned and mapped to four processors. The sparse grid (left) can be represented by its nodes in coordinate space. A Hilbert type space-filling curve, which fills the domain, is laid over the grid (right). Each node lies on the curve. Now we straighten the space-filling curve with the nodes fixed to the curve (bottom). The interval is cut into four sub-intervals assigned to one processor, each containing the same number of nodes.

the number of processors. Increasing the number of processors speeds up the computation accordingly. The parallel efficiencies are somewhat smaller than for the uniform refinement case, due to the imbalance in the tree of nodes. This is also the case for other parallel adaptive methods. Hence this parallelisation approach does perform very well, even in the range of higher number of processors 16 and 32, where a number of other strategies are not competitive.

## 6 Conclusion

We have proposed a parallelisation scheme for adaptive sparse grids with space-filling curves. Up to now, only standard sparse grids could be run on a distributed-memory parallel computer, based on the 'combination' technique for regular sparse grids. We have demonstrated the parallel efficiency of the approach for finite difference type of discretizations, both for regular and for adaptively refined sparse grids. This method can be applied to discretizations of many different PDEs and to other wavelet systems than the hierarchical basis.

## References

1. P. Bastian. Load balancing for adaptive multigrid methods. *SIAM J. Sci. Comput.*, 19(4):1303–1321, 1998.
2. H.-J. Bungartz. *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung*. PhD thesis, TU München, Inst. für Informatik, 1992.

| time | | processors | | | | | |
|---|---|---|---|---|---|---|---|
| nodes | 1/h | 1 | 2 | 4 | 8 | 16 | 32 |
| 81 | 4 | 0.03 | 0.04 | 0.05 | 0.07 | 0.11 | |
| 201 | 8 | 0.07 | 0.05 | 0.05 | 0.07 | 0.08 | 0.09 |
| 411 | 16 | 0.21 | 0.13 | 0.12 | 0.13 | 0.17 | 0.20 |
| 711 | 32 | 0.78 | 0.48 | 0.38 | 0.36 | 0.41 | 0.51 |
| 1143 | 64 | 2.60 | 1.49 | 1.06 | 0.93 | 0.92 | 1.14 |
| 1921 | 128 | 8.69 | 5.99 | 3.70 | 2.88 | 2.70 | 2.83 |
| 3299 | 256 | 39.3* | 20.7 | 13.8 | 9.62 | 7.79 | 7.32 |
| 6041 | 512 | 177* | 91.0 | 56.8 | 39.5 | 28.6 | 22.0 |
| 11787 | 1024 | 949* | 525 | 271 | 177 | 138 | 88.2 |
| 22911 | 2048 | | | 1280 | 761 | 660 | |

**Table 1.** Parallel execution times for adaptive sparse grids. A 3D convection-diffusion problem is solved and the solution times in seconds on Parnass2 are given.

3. G. Faber. Über stetige Funktionen. *Mathematische Annalen*, 66:81–94, 1909.
4. M. Griebel. The combination technique for the sparse grid solution of PDEs on multiprocessor machines. *Parallel Processing Letters*, 2:61–70, 1992.
5. M. Griebel. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. In *Proc. Large Scale Scientific Computations, Varna, Bulgaria*. Vieweg, 1998.
6. M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In P. de Groen and R. Beauwens, editors, *Iterative Methods in Linear Algebra*, pages 263–281. IMACS, Elsevier, 1992.
7. M. Griebel and G. Zumbusch. Adaptive sparse grids for hyperbolic conservation laws. In *Proceedings of Seventh International Conference on Hyperbolic Problems, Zurich*. Birkhäuser, 1998.
8. M. Griebel and G. Zumbusch. Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization. In J. Mandel, C. Farhat, and X.-C. Cai, editors, *Proc. Domain Decomposition Methods 10*, volume 218 of *Contemporary Mathematics*, pages 279–286, Providence, Rhode Island, 1998. AMS.
9. A. Harten. Multi-resolution representation of data: A general framework. *SIAM J. Numer. Anal.*, 33:1205–1256, 1995.
10. M. T. Jones and P. E. Plassmann. Parallel algorithms for adaptive mesh refinement. *SIAM J. Sientific Computing*, 18(3):686–708, 1997.
11. J. T. Oden, A. Patra, and Y. Feng. Domain decomposition for adaptive hp finite element methods. In *Proc. Domain Decomposition 7*, volume 180 of *Contemporary Mathematics*, pages 295–301. AMS, 1994.
12. M. Parashar and J. C. Browne. On partitioning dynamic adaptive grid hierarchies. In *Proceedings of the 29th Annual Hawai International Conference on System Sciences*, 1996.
13. S. Roberts, S. Kalyanasundaram, M. Cardew-Hall, and W. Clarke. A key based parallel adaptive refinement technique for finite element methods. In *Proc. Computational Techniques and Applications: CTAC '97*. World Scientific, 1998. to appear.

14. T. Schiekofer. *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen.* PhD thesis, Universität Bonn, Inst. für Angew. Math., 1998. to appear.
15. M. A. Schweitzer, G. Zumbusch, and M. Griebel. Parnass2: A cluster of dual-processor PCs. In W. Rehm and T. Ungerer, editors, *Proceedings of the 2nd Workshop Cluster-Computing*, number CSR-99-02 in Informatik Berichte. University Karlsruhe, TU Chemnitz, 1999.
16. S. A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSSR*, 4:240–243, 1963.
17. V. N. Temlyakov. Approximation of functions with bounded mixed derivative. *Proc. of the Steklov Institute of Mathematics*, 1, 1989.
18. C. Zenger. Sparse grids. In W. Hackbusch, editor, *Proc. 6th GAMM Seminar*, Kiel, 1991. Vieweg.
19. G. Zumbusch. Dynamic loadbalancing in a lightweight adaptive parallel multigrid PDE solver. In B. Hendrickson, K. Yelick, C. Bischof, I. Duff, A. Edelman, G. Geist, M. Heath, M. Heroux, C. Koelbel, R. Schrieber, R. Sinovec, and M. Wheeler, editors, *Proceedings of 9th SIAM Conference on Parallel Processing for Scientific Computing (PP 99)*, San Antonio, Tx., 1999. SIAM.