Think Unlimited and Compress Data Automatically

Maxime Schmitt, Philippe Helluy, Cédric Bastoul

Compas'2019 : Parallélisme / Architecture / Système
LIUPPA - IUT de Bayonne, France, du 24 au 28 juin 2019

Think Unlimited and Compress Data Automatically

Maxime Schmitt, Philippe Helluy, Cédric Bastoul

University of Strasbourg and Inria,
Laboratoire ICube — 300 bd Sébastien Brant
67412 Illkirch Cedex — France
{max.schmitt, helluy, cedric.bastoul}@unistra.fr

Abstract
Developing an application which, when unoptimized, consumes more memory resources than physically or financially available demands a lot of expertise. In this work, we show that with the right tools and language abstractions, writing such programs for a given class of applications can stay within reach of non-expert developers. We explore the potential of a compiler-based data layout transformation from dense array to a compressed tree data structure. This transformation allows easy application prototyping, provides compression and carries information that can be used with more advanced optimization, e.g., adaptive and approximate computing techniques. We are primarily targeting partial differential equation solvers and signal processing applications. We evaluate the compression ratio and error originating from this compressed representation. We suggest multiple exploration paths to produce an automatic adaptive code transformation with compressing capabilities from the multiresolution information produced during the transformation.

1. Introduction

Partial differential equation (PDE) solvers are a class of applications that may require a large amount of memory for large scale simulations. The usual discrete implementation of such solvers requires huge multidimensional domain representing the physical properties of interest. The simplest but most memory-consuming data structure implementation of the domain is a multi-dimensional array representing a regular n-dimensional mesh. To reduce the memory requirements, meshing techniques on a non-regular grid have been developed, but come at the price of a more complicated data management. For a maximum efficiency (in precision, time and space), they may be adaptive, allowing the non-regular grid to evolve during the execution of the simulation, refining the grid where the irregularities are located and coarsening it where the solution is smoother. Keeping track of these concepts and combining them requires a significant expertise. Moreover, implementing them is a time-consuming and error-prone development effort. We propose an abstraction allowing the developer to think about the data domain as a potentially infinite regular n-dimensional array that will be transformed by the compiler as a compressed tree data structure.
The tree data structure exploits the wavelet transformation (see Section 2.2) to extract multiresolution information and to compress data. We show that this transformation allows us to analyze the data features at multiple scales, leading to further optimization opportunities like automatic adaptive transformation. We motivate our choice of wavelet because of their properties for our application domain (see Section 2.3). We study a proof of concept wavelet compres-

sion program to motivate the usability of such transformation when data are too voluminous to fit in the computer memory. Lastly, we present complementary optimization to compression, e.g., using the tree structure efficiently or automatic adaptive code generation (see Section 3).

2. Automatic Sparse Data Transformation

In this section, we introduce our domain-specific data abstraction and the transformation to tree itself. We show that such transformation provides useful signal information which can be used to further optimize the application. We demonstrate how such transformation can be handled by the compiler as a source-to-source transformation in languages like `C` or `C++`.

2.1. Domain of Interest

Writing application dealing with a huge amount of data is not an easy task. For example, simulating physics experiments with large domain size in a reasonable amount of time usually requires a cluster of computers. The simulation data is scattered among all the computers and the simulation is solved locally by each node. Writing such applications is complex, hence we propose an abstraction allowing developers to write easy-to-maintain codes while still being able to leverage the hardware capacity at their disposal.

Our method resembles domain specific languages (DSL) by providing abstraction which simplifies developer's work. In DSLs, data structure and inter-node communications are usually not visible to the programmer, which has only access to the higher level view exposed by the DSL interface. In our method we transform an application that uses multi-dimensional array data structures to use a more appropriate sparse representation implemented internally as a tree. Sparsity is an important characteristic because it allows bigger datasets to be processed on a single node without requiring access to slow storage areas or allocating more nodes.

Another aspect of the transformation is related to adaptive techniques, i.e., targeting precise computation where it matters. We rely on the wavelet transformation to get relevant information about the shape of the data at multiple scales (see Section 2.2). We aim at using this information to generate an adaptive grid automatically.

Our goal is to provide a convenient way to think about the data, with simple n-dimensional arrays and let the compiler achieve the transformation to a sparse representation to be able to use this program in situation where dense data arrays cannot be used, e.g., the data is too voluminous for the available memory or it is too expensive to do all the computation at this fine scale. The compiler will replace the array access by the new structure access. It may also modify the execution order of the compute intensive kernel accessing the tree data structure to extract better performance from the underlying hardware.

2.2. Wavelets and Multi-Resolution Analysis

The wavelet transform is a routine tool for image and signal processing. Contrary to the Fourier transform [6] or Gabor transform [7], the wavelet transform provides both frequency and localization information at multiple scales. Figure 1 shows the frequency-time correspondence of the three transforms. The Fourier transform (Figure 1a) provides information about the frequency components of a signal but no time locality. Gabor introduced signal decomposition over *dictionaries* (set of functions) of time-frequency *atoms* (functions well localized in time and frequency). Figure 1b shows the Gabor transform of a signal where a window is translated in time and modulated in frequency, constructing the dictionary of time-frequency atoms, which are represented by rectangles in the figure. Each atom has a minimum surface area and the Heisenberg uncertainty principle links the temporal and frequency variance of the atom. This

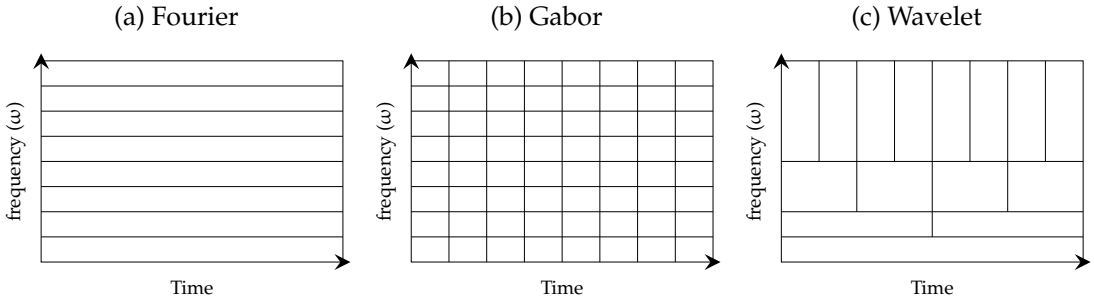

FIGURE 1 – Fourier, Gabor and Wavelet transformation time-frequency correlation.

imposes limits which forbids the construction of an atom with conjoint narrow time and frequency resolution. Hence, the size of the sliding window is a trade-off between the lowest observable frequency and the time precision. More than one Gabor transform is required to conduct a multi-scale analysis of the same data.
The wavelet transform is another way of constructing dictionaries. It is constructed from a *mother wavelet* Ψ which is scaled and translated to construct the dictionary [12, 5, 8]. Figure 1c shows the time-frequency relation of these atoms for the wavelet transform. The wavelet transform provides a good trade-off for the analysis of signal at multiple scales while retaining good time locality. In only one transformation we obtain the multiresolution data, removing the need for redundant Gabor transforms. This multi-scale analysis property may be exploited to achieve precise computation only on the relevant parts of the function.

2.3. Using a Wavelet With Appropriate Properties

A multiresolution analysis with convenient properties is crucial to extract useful information from a signal. Our goal is to achieve a good compression ratio, i.e. a good approximation in the wavelet basis, in order to allow program with a high memory footprint F_{mem} to be able to store its data on machines with a lower amount of physical memory $P_{mem} < F_{mem}$. Recent studies provide wavelet with various properties to apply in different scenarios [9, 4, 11, 1].
We choose the *coiflet* [4, 2] for its properties which are well suited for partial differential equations while providing a good compression ratio. It is defined by a scaling function Φ which carries the approximation of a signal at a given resolution, and a mother wavelet Ψ which carries the details necessary to increase the resolution of a signal approximation. The two functions have been crafted to yield the following properties:

Mother wavelet and scaling function form orthogonal basis This allows the separation of information at multiple scales. In addition, removing details does not change the total mass of the signal. It only decreases its energy. This is especially important in many physical applications.

The mother wavelet has N vanishing moments The first N terms of the Taylor's expansion of the signal do not contribute in the wavelet basis, i.e., functions that can be approximated by a polynomial up to the degree N will have a wavelet detail of zero. This allows for good compression of the data.

The scaling function has N vanishing moments This creates a function which is near interpolating. In other words, the corresponding wavelet coefficients are almost local samples of the signal.

The scaling function is near symmetric Symmetry allows an easier application of the dis-

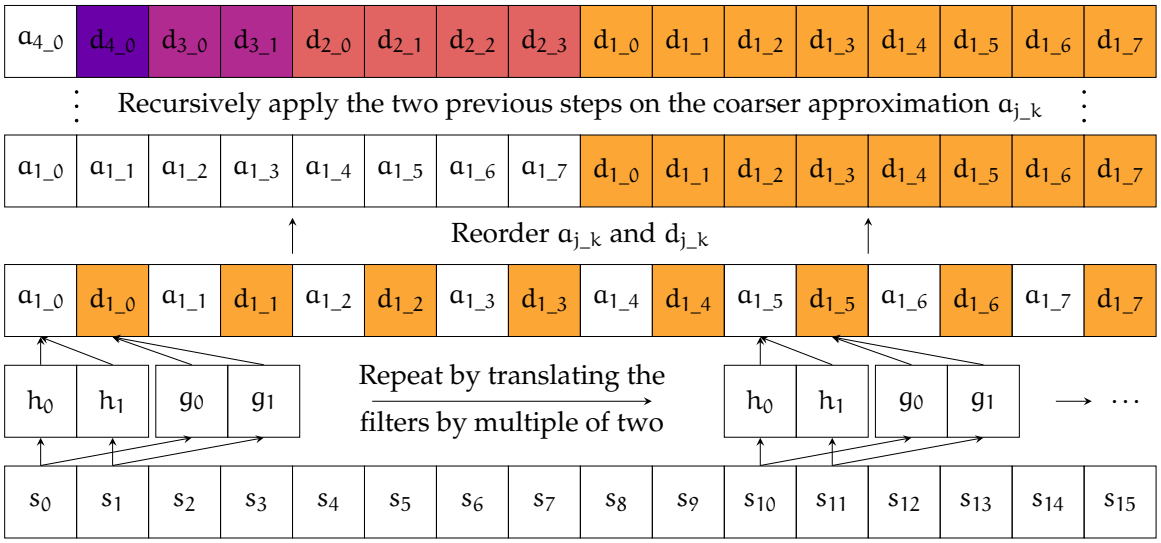

FIGURE 2 – The wavelet decomposition of an evenly sampled signal (s_i) with filters of size two. The algorithm consists of an application of filters h and g which respectively compute a coarser approximation (a_{j_k}) and the details lost by applying h (d_{j_k}) at the level j and a translation of $2k$. This step is usually followed by packing the coarser approximation and details together. These two steps are applied recursively until the number of remaining coarse approximation (a_{j_k}) is lower than the size of a filter.

crete wavelet transform on edge of discrete domains (e.g. images), where the data on the edge may be symmetrically replicated to avoid compression or distortion problems.

The scaling function has near linear phase Linear phase means that the wavelet transform will behave similarly at all frequency range.

2.3.1. Discrete Wavelet Transformation

The wavelet transform projects the signal to be analysed on the wavelets basis. The wavelets are constructed by scaling and translating the mother wavelet and scaling function. In order to have an efficient transformation, the scaling function can be defined using a discrete filter called *conjugate mirror filter*. This definition of the scaling function is $\Phi(t) = \sqrt{2}\sum_{n=-p}^{p-1} h[n]\Phi(2t-n)$, where h is a discrete filter of size $2p$. The wavelet function Ψ can also be defined in the same way and we can prove that its filter is then $g[n] = (-1)^{1-n}h[1-n]$ for ensuring orthogonality. These filters allow for a practical implementation of the wavelet transform with a complexity of $O(n)$ for data of size n.

The fast orthogonal wavelet transform consists in consecutive application of the filters h and g defining the scaling function and the mother wavelet [10]. This transformation is depicted in Figure 2. The g filter creates the local details, while the h filter creates a local approximation of the function free of the details. This step is repeated on the approximation generated by h to continue the analysis on a larger scale. The reverse wavelet transform works from top to bottom, recombining details and approximations to reconstruct the function.

2.4. Dense to Sparse Data Transformation

We propose a data layout transformation based on the wavelet transformation to store a dense data set in a sparse data structure. This transformation yields a good compression ratio if the

```
1 #define N 2048
2 #pragma sparsify<coif> foo bar
3 float foo[N], bar[N];
4 for (size_t i = 0; i < N; ++i)
5   foo[i] += bar[i];
```

The annotation at line 2 instructs the compiler to change the data representation of the arrays `foo` and `bar` to use the Coiflet. The data structure allocation, accesses and automatic compression are handled by the compiler.

FIGURE 3 – Example of code annotation to perform the wavelet data layout transformation

data is smooth enough. Mathematically, the transformation is limited to functions in $L^2(\mathbb{R})$, i.e., squared integrable functions $\int |f(x)|^2 dx < \infty$, hence we restrict ourselves to floating point arithmetic without infinite values.
From a programmer point of view, the approach works as follows. The developer writes the application using a simple dense data representation, i.e., an n-dimensional array, but instructs the compiler to use the sparse wavelet representation instead. Figure 3 sketches the implementation of such transformation. The user places an annotation inside the source code to use the capabilities of our data-structure. For this example transformation, no other transformation than the data layout is applied. Taking full advantage of the new layout may require a new computation ordering. This and further optimization opportunities are discussed in Section 3.

2.4.1. Early Compression Evaluation

We implemented a prototype to evaluate the potential compression ratio achievable on 1-dimensional data arrays. We implemented the coiflet wavelet transform using filter of size 6. Figure 4 shows the decomposition of a signal composed of a sine and exponential into a graded tree representation. We can achieve a 97% compression rate of this test signal with a maximum error of 0.2% and mean error of 0.002% when comparing the approximated and the original function values. The destructive compression occurs when a detail which is really close to zero is actually set to zero to yield more compression because we don't store the details anymore. A sharp portion of the function, at each scale, has a color closer to blue and smooth ones closer to yellow. We can see from this representation that large portions of the tree can be compressed and that its shape gives us information about the function at multiple resolutions.

2.4.2. Early Overhead Evaluation

We used our prototype (see Section 2.4.1) to measure a $7\times$ slowdown between the time to access all the elements of the initial array, and the time to compress the whole array plus the time to access all its elements including decompression. This slowdown remains constant, disrespectfully of the array size. We consider it as quite encouraging since no specific optimization has been performed and considering the high potential compression ratio.

3. Leveraging Wavelet Sparse Information and Representation

Using a sparse representation based on the wavelet transform provides multiresolution information and a compact representation for the data. However, exploiting this representation efficiently and extracting algorithmic optimization is a real challenge. The automatic translation from dense to sparse representation (see Section 2.4) is the first step towards efficient automation of the data transformation.
We are in the process of evaluating this abstraction on a fluid dynamic simulation using the Lattice Boltzmann method [3]. This application is written in C using regular arrays with annotation instructing the compiler to use our special data structure (see Section 2.4). The computa-

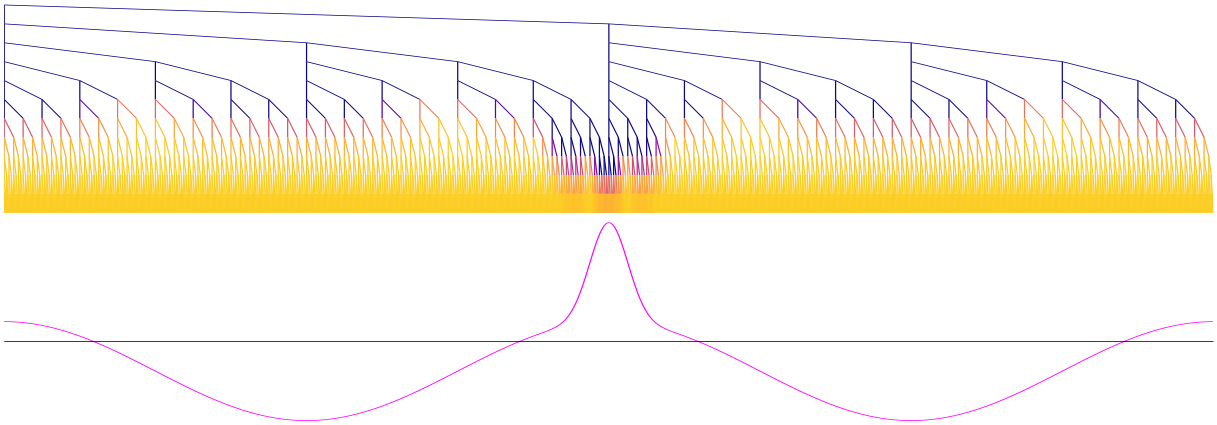

FIGURE 4 – Tree representation of the wavelet details from the signal present at the bottom. The edges of the tree is colored in a scale ranging from light yellow to dark blue corresponding respectively to low and high detail coefficients. Each level of the tree corresponds to a frequency level. The high frequency are present at the leaf and the low frequency at the root. The detail coefficient at a given scale has a high value if the signal is sharp or low if it is smooth.

tion is done at the finest scale (same as the dense arrays) while the data structure transparently compress the data. Our goal is to evaluate the following optimization opportunities:

Loop ordering optimization A tree data structure is well suited for representing the wavelet transform. However the data traversal in the original loop is not likely to correspond to the new tree data layout, resulting in a bad data locality. This may be improved by applying loop tiling with appropriate tile sizes and tile ordering along with the data layout transformation.

Multiscale optimization Use the information provided by the wavelet transformation to generate a code that will do the computation at the scale where the details are high enough. An example algorithm to use the wavelet tree efficiently would be to firstly refine the tree to avoid losing details, solely achieve the computation for the tree leaves and finally compress the tree to erase the leaves with low details.

Overhead, error and compression ratio evaluation The overhead of managing the tree with the mentioned techniques has to be evaluated along with techniques to bound or predict the error coming from the algorithm used during the compression phase.

4. Conclusion

Writing an application with abstractions allows developers to focus their efforts on the problem. We propose a compiler-assisted data layout transformation which provides transparent compression and multi-scale information of square integrable functions exploiting wavelet transform. Provided the data is smooth enough, e.g., application in PDE, image and signal processing, we allow the developers to think about their data as dense multi-dimensional arrays. We show that this technique may achieve high compression ratio and allow the data to be stored in main memory, even when the dense array may not fit. The wavelet transformation behind the data transformation provides a multiresolution analysis which opens a wide range of optimization opportunities. Ongoing investigation aim at achieving fast access to compressed data and at exploiting the adaptive opportunities offered by the transform.

Bibliographie

1. Battle (G.). – A block spin construction of ondelettes. part i: Lemarié functions. *Communications in Mathematical Physics*, vol. 110, n4, Dec 1987, pp. 601–615.
2. Černá (D.), Finěk (V.) et Najzar (K.). – On the exact values of coefficients of coiflets. *Central European Journal of Mathematics*, vol. 6, n1, Mar 2008, pp. 159–169.
3. Chen (S.) et Doolen (G. D.). – Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, vol. 30, n1, 1998, pp. 329–364.
4. Daubechies (I.). – Orthonormal bases of compactly supported wavelets. *Communications on pure and applied mathematics*, vol. 41, n7, 1988, pp. 909–996.
5. Daubechies (I.). – *Ten Lectures on Wavelets*. – Philadelphia, PA, USA, Society for Industrial and Applied Mathematics, 1992.
6. Dym (H.) et McKean (H.). – *Fourier Series and Integrals*. – Academic Press, 1972, *Probability and mathematical statistics*.
7. Gabor (D.). – Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, vol. 93, n26, November 1946, pp. 429–441.
8. Grossmann (A.) et Morlet (J.). – Decomposition of hardy functions into square integrable wavelets of constant shape. *SIAM journal on mathematical analysis*, vol. 15, n4, 1984, pp. 723–736.
9. Haar (A.). – Zur Theorie der orthogonalen Funktionensysteme. *Mathematische Annalen*, vol. 69, n3, Sep 1910, pp. 331–371.
10. Mallat (S. G.). – A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no7, 1989, pp. 674–693.
11. Meyer (Y.). – Principe d'incertitude, bases hilbertiennes et algèbres d'opérateurs. *Séminaire Bourbaki*, vol. 28, 1986, pp. 209–223.
12. Stéphane (M.). – *A Wavelet Tour of Signal Processing - The Sparse Way*. – Elsevier, 2009.