Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 491–494

ISBN 978-83-60810-22-4

Implementation of Movie-based Matrix Algorithms on OpenMP Platform

Dmitry Vazhenin
Graduate School Department of Information Systems
University of Aizu
Tsuruga, Ikki-machi, Aizu-Wakamatsu, Japan
Email: d8052102@u-aizu.ac.jp

Alexander Vazhenin
Graduate School Department of Information Systems
University of Aizu
Tsuruga, Ikki-machi, Aizu-Wakamatsu, Japan
Email: vazhenin@u-aizu.ac.jp

***Abstract*—The convenience and programmer's productivity are the main point of visual programming systems and languages. From the other side, the parallel programming is mainly focused on reaching the high performance by optimization of executable code. The Movie-based Programming is based not only on the introduction of special symbols and images with semantic support, but also on a series of images that can present dynamical features of algorithms. The presented paper describes a technique of OpenMP parallelization of Movie-based algorithms in order to obtain the suitable program performance. The results of numerical experiments are also presented showing applicability of the proposed technique including implementation, code validity checking and performance testing.**

***Index Terms*—Visual Programming, Movie-based programming, Matrix Computing, Parallel Programming, OpenMP Platform.**

I. Introduction

ORIGINALLY, the Visual Programming Languages (VPL) are oriented to increase mainly the programmer's productivity by operating with visual expressions, direct manipulating visual information as well as supporting visual interactions. The most of modern applications of this kind include a big variety of attractive multimedia functions with icons, pictures, animations, sound and other multimedia components allowing reliable understanding as well as effective dialogue with the complex objects. Visual programming languages and tools may be classified according to the type and extent of visual expression used, into icon-based languages, form-based languages and diagram languages as shown in reviews [1], [2]. They provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program.

Usually, the parallel programming and algorithm design are focused on reaching the high performance by optimization of executable code as shown, for example, in [3]-[4]. This may contradict with the VPLs in which convenience and programmer's produtivity are the main point of system requirements. For example, a graphical tooltkit is described in [5] consisting of exploratory tools and estimation tools which allow the programmer to navigate through complex distributions and to obtain graphical ratings with respect to load distribution and communication. The toolkit has been implemented in a mapping design and visualization tool which is coupled with a compilation system for the HPF [6] predecessor Vienna Fortran. Since this language covers a superset of HPFs facilities, the tool may also be used for visualization of HPF data structures. The GASPARD (Graphical Array Specification for PARallel and Distributed computing) is a visual programming environment devoted to the development of parallel applications [7]. Task and data parallelism paradigm are mixed in GASPARD to achieve a simple programming interface based on the printed circuit metaphor.

The Movie-based Programming is our approach for promoting high-level language constructs introducting not only special symbols and images with semantic support, but also on a series of images that can present dynamical features of algorithms. The usage of this approach for numerical solution of some linear algebra problems allows to generate automatically rather effective sequential executable C-code [8]-[10]. OpenMP, a portable programming interface for shared memory parallel computers, was adopted as an informal standard in 1997 by computer scientists who wanted a unified model on which to base programs for shared memory systems [11]. The usage of OpenMP offers a comprehensive introduction to parallel programming concepts. The goal of this work is to develop a technique of OpenMP parallelization of Movie-based algorithms in order to obtain their high performance.

The rest of the paper is organized as follows. In Section 2, we discuss a concept of the Movie-based Programming including component description and definitions. The third section describes code generation process and adaptaion it to the OpenMP environment. In Section 4, demonstrates the least-squares polynomial curve fitting as an example of the usage of our approach. The last section contains conclusion.

II. Movie-Based Components and Definitions

The Movie-based representation of computational methods and algorithms is based on a correspondence between algorithmic movie frames and problem solution steps. Accordingly, each frame should visualize/animate a corresponding step of a program/algorithm execution. Within the frame, structures are shown as a statical images representing parameterized sets of nodes which can be connected by links in 4D space-time. The structure is any geometrical construction in a 3D space. Let us

consider the programming process as a specification of the following statements: *WHEN*-statement, *WHERE*-statement and *WHAT*-statement.

WHEN-statement. As was mentioned above, a *frame* is an image representing dynamical features of an algorithm at a particular time step. So, sequences of frames are observable as an animation and can be composed and visually debugged. A computational step is visualized as a combination of visual symbols within a frame. A set of frames joined into an animated sequence – i.e. a visualization of a computation on a structure according to the traversal schemes and the computational formulas for the frame – is called an "algorithmic film" or just "film" that is a combination of $\mathcal{S}$ as a collection of spatial structures, $\mathcal{D}$ as a set of variable declarations, $\mathcal{M}$, a collection of metaframes.

A *metaframe* is a special object representing a set of rules and parameters which are meant to specify how frames should be produced (visualized) in a film, and, how they should be implemented in an executable program. The two main types of metaframes, *single* and *episode*, are used to specify single or multiple algorithmic steps respectively. Any metaframe is a combination of $\mathcal{T}$ as a set of traversal schemes for node activation on structures in $\mathcal{S}$; $\mathcal{C}$ as a set of control-flow formulas exploited in the frame-generation process, and $\mathcal{F}$ as a set of computational formulas to be performed on nodes affected by schemes in $\mathcal{T}$;

WHERE-statement. Rather often, the data structures in applications can be regular ordered sets of elements presented as 1D, 2D or 3D structures where structure nodes contain operable objects. So, sets of elements involved in computations can be related to the sub-domain nodes of these structures. Any structure has the attributes shown in (Figure 1) including **Name**, **Dimension**, **Parameters** to define structure sizes (N,M for rows and columns respectively), **Structure Nodes** as well as a set of **Structure Variables** ($\widetilde{A}$) declared to store instances of data in nodes, a set of sub-domain variables ($\widetilde{R}$) and domain variables (R_Δ). It includes also a set of **Control Lines** which are used to refer to spatial placements and domains.

A sub-domain is a set of nodes which coordinates are satisfying to a system of constraints $\Omega = \{(i,j)|H_1 \leq i < H_2 \wedge V_1 \leq j < V_2 \wedge P_1(i,j) \wedge P_2(i,j) \wedge \ldots \wedge P_n(i,j)\}$, where (i,j) - coordinates of nodes, H_1,H_2,V_1,V_2 - positions of appropriate control lines or structure bounds and predicates $P_k(i,j)$ - special conditions (they are used to specify the sub-domain shape). Accordingly, a domain $\Delta = \Omega_1 \bigcup \Omega_2 \ldots \bigcup \Omega_n$ can also be defined as a composition of sub-domains Ω_i.

Any domain needs to have a special attribute to be visually distinguishable from other domains. This attribute marks all domain elements by color, shape, size, etc. Usually, we are using a color attribute that allows the user to operate with parametric relation-based specification.

WHAT-statement. All frame attributes can change their parameters by assigment operations on them. To specify these changes, the user should assign operators on a metaframe attributes that have to be implemented during frame processing.

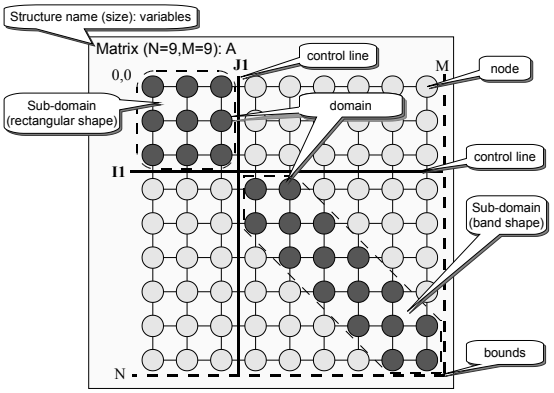


Fig. 1. Attributes and features of matrix structure and its sub-domains.

We distinguish two types of these operators: 1) *Computational formulas* to specify operations on node domains, 2) *Control-flow formulas* to define frame transitions (control line movements and episode conditions).

III. MP-Templates and Code Generation

As was shown in previous section, *a domain* Δ consists of elementary traversal schemes that parametrically enumerate elements arranged in a particular shape (dot, row, column, rectangle, triangle, band, etc). These schemes are called Movie/Program template or *MP-templates* each of which is a set of structure nodes which coordinates are satisfying to a system of constraints Δ, and considered as a complete subpart of MP-program if it has C-formula attached. C-formula is defined as a subprogram containing a sequence of arithmetical and/or logical expressions.

By using appropriate types of metaframes, domain configurations and corresponding formulas, it is possible to specify an algorithm visually on a defined structure. Such an algorithm can be debugged on a structure of a small size appropriate for understanding algorithmic steps; however, the resulting code will be operable on a structure with a potentially unlimited size defined by corresponding parameters. The code generation is based on special template code snippets performing scanning steps on predefined elementary sub-domains: row, column, diagonal, triangle, etc, which are combined into scanning templates of an arbitrary complex domain. These template programs are used both 1) for visual demonstration of effected structure nodes within domains on a movie frame, and 2) for computation by applying formulas to appropriate elements.

The generation of executable code is realized using relation-based parameterized specifications of metaframes with domains, control lines and formulas as well as embedded transformation rules as shown in [10]. To obtain the parallel executable code, a set of template samples in C language for an OpenMP target platform was developed. This development was realized by embedding OpenMP Directives into original templates with authomatic substitution of the metaframe attributes. As it is demonstrated below, this simple transformations allowed to obtain relatively effective parallel code.

IV. Implementation on OpenMP Environment

A. Movie-based Least-Squares Optimization Method

The Least-Squares Optimization method is a mathematical procedure for finding the best-fitting polynomial curve $f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m$ along a given set of points $(x_1, y_2), (x_2, y_2) \ldots (x_n, y_n)$ by minimizing the sum of the squares of the differences between a curve and points. The coefficients of a curve $a_0, a_1, \ldots, a_m$ are found by solving the following SLAE $A\vec{x} = \vec{b}$:

$$\begin{array}{rcl} a_0\sum_{i=1}^{n}1+a_1\sum_{i=1}^{n}x_i+\ldots+a_m\sum_{i=1}^{n}x_i^m & = & \sum_{i=1}^{n}y_i \\ a_0\sum_{i=1}^{n}x_i+a_1\sum_{i=1}^{n}x_i^2+\ldots+a_m\sum_{i=1}^{n}x_i^{m+1} & = & \sum_{i=1}^{n}x_i y_i \\ \ldots & & \\ a_0\sum_{i=1}^{n}x_i^m+a_1\sum_{i=1}^{n}x_i^{m+1}+\ldots+a_m\sum_{i=1}^{n}x_i^{2m} & = & \sum_{i=1}^{n}x_i^m y_i \end{array} \quad (1)$$

To generate the SLAE (1) it is necessary to calculate coefficients as sums that have arguments with degree parameter related to the position of a corresponding sum operation. Fig. 2 demonstrates a part of this algorithm related to the matrix builder. This episode is to fill up the matrix elements with the corresponding numbers from the left part of the SLAE.

There are three 2D grid structures defined. Structures "vector1" and "vector2" have a colored domain with the single sub-domain of row type defined to emphasize the first row which corresponds to the x coordinate of all points. Matrix structure "Matrix" has a colored domain with the single sub-domain of the minor-diagonal type moving from the top-left corner to the bottom-right corner. According to corresponding transitional expressions this frame-to-frame movement is controlled by two control lines attached: I_1 and J_1. This episode is continued until I_1 has reached the bottom border acoording to the specified conditional expression ($I_1 < N$).

This specification allows the generation of executable source code which is presented at the bottom of Fig. 2. It is possible to see how the OpenMP operators are embedded into the C code generated. The rest of the algorithm includes vector $\vec{b}$ building and SLAE solving metaframes which have been omitted due to the space limitations.

B. Numerical Experiments

The first type of experiments were implemented in order evaluate the quality of the generated source code text in comparison with a sample solution taken from the tutorial [12] by the automatic code validator "splint" [13]. The other experiments evaluated the resulting code performance. The same code from the previous test have been used as a sequential sample to run on the single CPU. Then, OpenMP templates have been used to generate parallel sample codes. Example of results achieved in this experiment are demonstrated on Figure 3 with a dataset for n=10,000,000.

The left-side graphs in each dataset shows dependence between the time elapsed by each running code and the matrix size m (1). The right-side graphs in each dataset show speedup of parallel code in respect to the sequential one. Testing platform: SMP server with four Intel Xeon 4-core X5550 2.67GHz (16 cores totally), with 12GB of memory, running Ubuntu 10.04 x86_64 (server edition). The results demonstrate that both generated sequential and parallel programs can reach a suitable performance.

V. Conclusion

The Movie-based programming environment is presented using a concept of metaframes controllable by a special interface panels for specifying algorithmic movie-frames and spawning the automatic code generation. This includes the design of new visual symbols and the introduction of an automatic template generation technique supporting compact specification of computational expressions, and allowing an essential advance of the library of templates oriented to specify a variety of numerical matrix algorithms as well as parallel programming platforms. Visual semantic and syntactic rules have been presented defining domains with various shapes on computational structures, and various operations on those structures. The adaptation of a Movie-based program to parallel mode is implemented by modifying the shape-scanning templates according to the OpenMP programming rules. This simple modification allowed to generate a parallel code with relatively good quality. Applicability of the technique and environment has been demonstrated through the Least-squares Curve Fitting Method including implementation, code validity checking and performance testing.

References

[1] M. Burnett, Visual Programming, *Wiley Encyclopedia of Computer Science and Engineering*, John Wiley & Sons Inc., Hoboken, 1999.

[2] , Ph. T. Cox, Visual Programming Languages, *Wiley Encyclopedia of Computer Science and Engineering*, John Wiley & Sons Inc., Hoboken, 2008.

[3] R. Rabenseifner, Optimization of collective reduction operations, *LNCS*, Springer-Verlag, Vol. 3036, 2004, 1-9.

[4] J. Pjesivac-Grbovic, and T. Angskun, and G. Bosilca, and G. E. Fagg, and E. Gabriel and J. Dongarra, Performance analysis of MPI collective operations, Cluster Computing, Kluwer Acad. Publ., Vol. 10, No. 2, 2007, 168-179.

[5] S, Grabner, and R. Koppler and J. Volkert, *Visualization of Distributed Data Structures for HPF-like Languages*, Technical Report, Johannes Kepler University Linz, 2005.

[6] *High Performance Fortran (HPF)*, http://www.netlib.org/hpf/.

[7] Fl. Devin, and P. Boulet, and J. L. Dekeyser and Ph. Marquet, "GASPARD - a Visual Parallel Programming Environment", *Proc. of the Int. Conf. on Parallel Computing in Electrical Engineering (PARELEC'02)*, 2002, 145-150.

[8] D. Vazhenin, A. Vazhenin and N. Mirenkov, "Movie-based Multimedia Environment for Programming and Algorithms Design", *LNCS*, Springer-Verlag, Vol. 3333, No. 3, 2004, 533-541.

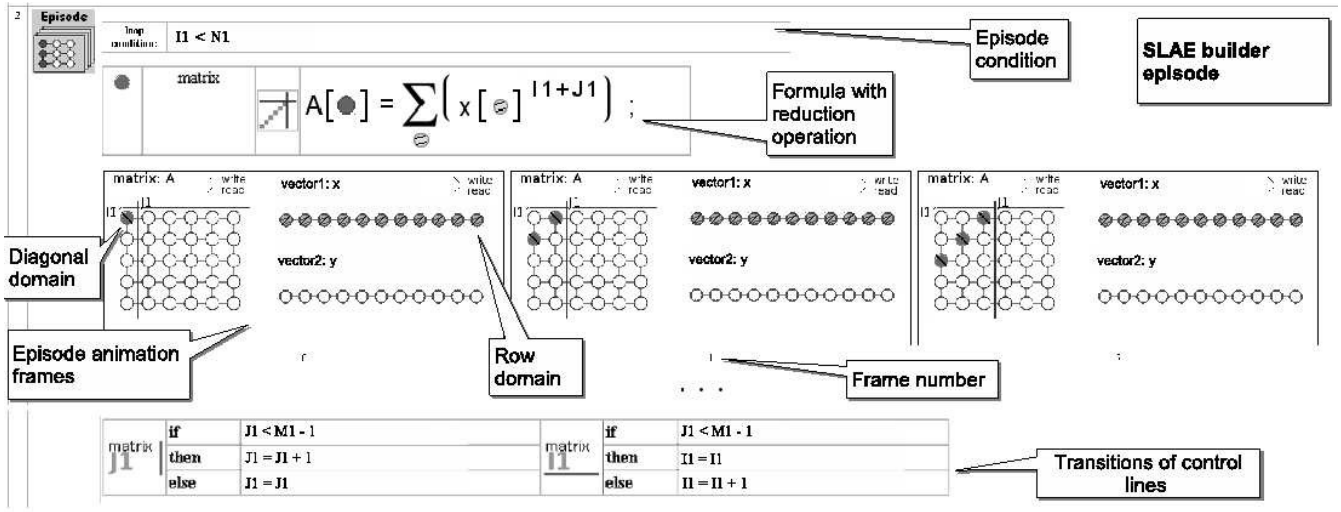
[9] D. Vazhenin, A. Vazhenin and N. Mirenkov, "Movie-based templates for linear algebra problems", *Int. Jour. of Comp. Sci. and Network Security*, Vol. 7, No. 1, 2007, 378-385.

[10] D. Vazhenin, A. Vazhenin, "MP-templates Operating Toolkit in Movie-based Programming", *Proc. of Japan-China Workshop on Frontier of Computer Science and Technology*, Nagasaki, Japan, 2008, 67-73.

[11] *OpenMP: Simple, Portable, Scalable SMP Programming*, http://www.openmp.org.

[12] T. Veerarajan, and T. Ramachandran *Numerical Methods: With Programs In C*, Tata McGraw-Hill, 2005.

[13] *Splint: statical C code validator* http://www.splint.org.




```
main() {
...
   /* episode metaframe number 2 */
   while(matrix_I1<matrix_N1) { // episode begin
    /* Domain color:rgb[255,0,0] */
    /* Subdomain Diagonal2 template begin */
    _St_Row=matrix_I1; _St_Col=0; _Fin_Row=matrix_N;
    _Fin_Col=matrix_J1; _Step_Row=1; _Step_Col=1;

    #pragma omp parallel for private(_Scan_I,_ScanF_I) default(shared)
    for(_Scan_I=0; _Scan_I<_Fin_Row-_St_Row &&
        _Scan_I<_Fin_Col - _St_Col; _Scan_I+=_Step_Row)
    {
       _ScanF_I=_Scan_I+_St_Row;
       _ScanF_J=_Fin_Col-_Scan_I-1;
       A[_ScanF_I][_ScanF_J]=fc_reduce_sum1(x, _control_data);
    }
    /* Subdomain Diagonal2 template end*/
    /* moving control lines: */
    If(matrix_J1<matrix_M1-1) matrix_J1=matrix_J1+1;
    If(matrix_J1<matrix_M1-1); else matrix_I1=matrix_I1+1;

    } // episode end
...
}
```

```
/* generated temporary template for reduction sum */
double fc_reduce_sum1(double **x, _ControlDataP _control_data) {
 #include "fillmdefs.h"
 double result=0.0;
 /*** Mirroring main control lines and structure size via _control_data ***/
 int vector1_M=*_control_data->size[0];
 ...

 #pragma omp parallel reduction(+ : result)
 {
    /* Domain color: rgb[255,0,255] */
    /* Subdomain Row template begin */
    _St_Row=0; _St_Col=0; _Fin_Col=vector1_M;
    _Step_Row=1; _Step_Col=1; _Scan_I=_St_Row;

    #pragma omp parallel for private(_Scan_J,_ScanF_J) default(shared)
    for(_Scan_J=_St_Col; _Scan_J<_Fin_Col; _Scan_J+=_Step_Col) {
        _ScanF_I=_Scan_I;
        _ScanF_J=_Scan_J;
        result+=(pow(x[_ScanF_I][_ScanF_J],(matrix_I1+matrix_J1)));
    }
     /* Subdomain Row template end*/
 }

 return result;
}
```

Reduction formula

Fig. 2. Movie-based program of least-squares method and an excerpt of generated code

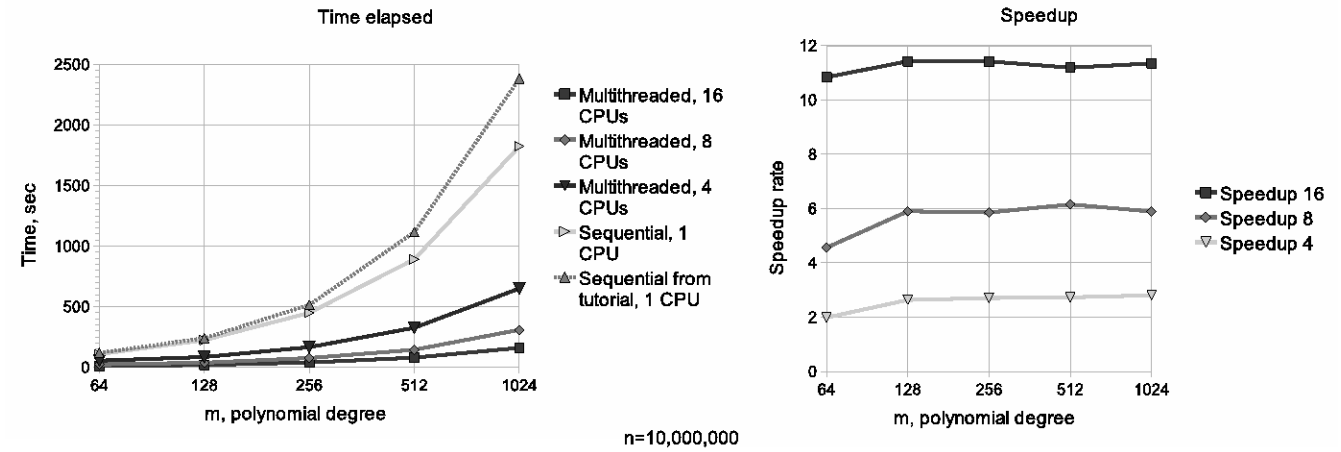


Fig. 3. Movie-based program of least-squares method and an excerpt of generated code