Procedia Computer Science

Volume 29, 2014, Pages 102–112

ICCS 2014. 14th International Conference on Computational Science

Blood Flow Arterial Network Simulation with the Implicit Parallelism Library SkelGIS

Hélène Coullon[1],[3], Jose-Maria Fullana[2], Pierre-Yves Lagrée[2], Sébastien Limet[1], and Xiaofei Wang[2]

[1] Univ. Orléans, INSA Centre Val de Loire, LIFO EA 4022, F-45067 Orléans, France
helene.coullon@univ-orleans.fr, sebastien.limet@univ-orleans.fr
[2] CNRS & UPMC Univ Paris 06, UMR 7190,
Institut Jean Le Rond d'Alembert, Boîte 162, F-75005 Paris, France
fullana@ida.upmc.fr, pierre-yves.lagree@upmc.fr, xfwang0219@gmail.com
[3] Antea Géo-Hyd, 101 rue Jacques Charles, 45160 Olivet, France

Abstract

Implicit parallelism computing is an active research domain of computer science. Most implicit parallelism solutions to solve partial differential equations, and scientific simulations, are based on the specificity of numerical methods, where the user has to call specific functions which embed parallelism. This paper presents the implicit parallel library SkelGIS which allows the user to freely write its numerical method in a sequential programming style in C++. This library relies on four concepts which are applied, in this paper, to the specific case of network simulations. SkelGIS is evaluated on a blood flow simulation in arterial networks. Benchmarks are first performed to compare the performance and the coding difficulty of two implementations of the simulation, one using SkelGIS, and one using OpenMP. Finally, the scalability of the SkelGIS implementation, on a cluster, is studied up to 1024 cores.

Keywords: Implicit parallelism, network simulations, PDEs, Blood flow

1 Introduction

A scientific simulation program is an imitation of a process over time on a computer. Most of the time, the real phenomena is modeled by a system of partial differential equations which are time- and space-dependent (PDEs). In almost all cases, it is not possible to directly solve analytically these equations. Hence, approximated solutions of the system are computed using numerical methods [14, 15] which discretize time and space. In a program, the discrete representation of space, called a *mesh*, is implemented by a data structure which models the connectivity between space elements. Time discretization, on the other hand, is translated to a main loop for time iterations. Three numerical methods with space discretization are more popular. The first one, based on the derivation from Taylors polynomial, is called *finite difference method*. In this

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2014

doi:10.1016/j.procs.2014.05.010

method the spatial domain is discretized by a regular mesh, typically a Cartesian grid. In the second method, called *finite volume method*, the domain can be discretized by a general mesh where elements are called volume controls. This method computes the averaged value of the exact solution on each mesh element by integrating the given system on volume controls. The third resolution method is called *finite element method*. The exact solution is approximated by a continuous and piecewise polynomial function and the domain is often discretized by a triangulation. Each time iteration applies the scheme, obtained by the resolution method, to the mesh. The type of scientific simulations we are interested in can be represented by the explicit scheme

$$\{U_{t-1}(x), U_{t-1}(y); y \in N(x)\} \longmapsto U_t(x), \tag{1}$$

where x represents an element of the mesh (i.e. discretized space domain), $U_t(x)$ is the set of quantities to compute for element x at the time iteration t, $N(x)$ is the neighborhood of x required to compute $U_t(x)$ in the mesh. The precision of a simulation is defined by its order which gives information on needed neighborhood. In computer-science, such a computation is called a *stencil*, and this domain is currently arousing a great deal of interest. The last important point concerning scientific simulations is the concept of physical border of the domain. In a real-world process, the domain is not limited, however, in a program, the discretized mesh has to be finite. Thus, specific elements have to be created, to define the behavior of the physical border of the mesh.

Complexity of scientific models and precision of data to compute in simulations are growing. Moreover, access to super-computers is becoming easier with a growing number of clusters over the world. As a result, it becomes significant for scientists of all domains to write parallel programs. For this reason, and because of lack of time and resources to get efficient parallel programs, implicit parallelism research is an active domain of computer-science. To be used, an implicit parallelism solution has to find a good abstraction level for the user. Indeed, if the solution is too generic, it works for most scientific problems, but it could be difficult to use. On the other hand, if a solution is too specific, it could be too limited for most users. In this paper is presented the implicit parallelism library SkelGIS [6, 7]. This library is specific to scientific simulations which can be represented by the scheme (1). However, this simulation class is very large, as most scientific simulations are solved with explicit numerical schemes.

The implicit parallelism library odeint [1] solves ordinary differential equations and proposes implicit parallelism on GPUs with CUDA. PETSc [2] solves partial differential equations and proposes implicit parallelism for GPUs, with CUDA, CPUs, with MPI and pthreads programming, and hybrid systems. The main difference with SkelGIS is the view of the user on the simulation, and his programming freedom. SkelGIS does not provide predefined tools to solve differential equations, as for example *interpolate* or *jacobi* functions etc. The abstraction level proposed by SkelGIS is closer to the OP2 framework [16] and the domain specific language Liszt [9]. OP2 and Liszt both are implicit parallelism solutions to solve partial differential equations based on unstructured-meshes solutions. In SkelGIS, the user has the total control on the produced code. However, all complex data structures, optimizations and parallelizations are hidden from the user through four macroscopic concepts: *distributed data structures*, *data mapping*, *applier* and *programming interface*. Finally, SkelGIS can be compared to generic libraries as for example STAPL [5] which is a parallel version of the C++ standard library (STL). However, SkelGIS is specific to scientific simulations, while STAPL is as generic as the STL.

The four concepts of SkelGIS have already been applied to the case of two-dimensional regular meshes. Both heat equation and shallow-water equations have been solved using SkelGIS [6, 7]. Efficiency of SkelGIS programs were convincing compared to the equivalent MPI

codes, and with a smaller coding effort, according to Halstead metrics [11]. SkelGIS aims to apply its four concepts to different kinds of simulations and to different kinds of meshes. This paper deals with the case of network simulations which highlights difficulties which do not occur in a simulation on a two-dimensional regular mesh. In a network simulation, the domain is divided in two different kinds of elements with two different behaviors (different schemes): nodes and edges. A network also represents the connectivity between different elements of the domain, as does a mesh. A network is typically used for arterial or vein simulations, road or rail traffic simulations, water-flow or pollutant transfer simulations etc. A network can be represented as a graph and in most cases as a directed acyclic graph (DAG). The physical border of a network is simulated by the specific behaviors of roots and leaves of the DAG. As far as we know, no specific implicit parallelism solutions exist to solve network simulations. Libraries like PETSc, which implement sparse matrices, could be used since a network can be represented by a sparse matrix. However, two different schemes have to be computed, then multiple kinds of sparse matrices, for nodes and edges, have to be managed by the user. On the other hand, using OP2 and Liszt, an unstructured mesh has to be created by the user, however a network is not an unstructured mesh and is not composed of faces and cells. Thus, it seems impossible to define a network with those solutions. The work presented in this paper offers an intuitive way to write network simulations and to obtain efficient parallel programs. The paper studies implementation of a complex network simulation using SkelGIS and is organized as follow. Concepts of the SkelGIS library as well as details on the specific case of networks are explained in the next section. Then, the blood-flow arterial simulation and its numerical resolution are detailed. Performance results are compared to an OpenMP version of the same simulation, and efficiency on a big network is evaluated on a cluster. Finally, conclusion and perspectives on this work are given.

2 SkelGIS implicit parallelism library

The SkelGIS library is an implicit parallelism solution for scientific simulations. It aims to offer a transparent access to parallel computing. Niklaus Wirth's aphorism [18] "Program = Algorithm + Data Structure" (1) can be transposed to SPMD parallel programs (Single Program Multiple Data) on distributed memory architectures: "Parallel program = Distributed data + Algorithm + communications" (2). SkelGIS generates MPI parallel programs of type (2), however, it aims at providing a sequential programming style of type (1). Figure 1 illustrates the SkelGIS principles that relies on four concepts named *DDS*, *DPMap*, *AP* and *PI*. *DDS* is a distributed data structure which manages the mesh and its connectivity and distributes automatically the mesh among processors. Most of the efficiency of SkelGIS relies on the DDS. *DPMap* is a data mapping. Each of its instantiation represents data of the simulation (quantities it uses) and its mapping on the DDS. *AP* is an applier. It is used to apply a sequential user function, called an *operation*, to a set of DPMaps. An applier also transparently proceeds MPI communications between processors. Finally, *PI* represents the programming interface of SkelGIS. *PI* is used by the programmer to navigate through the data, read and update them. This interface is based on iterators and specific functions to access the neighborhood of mesh's elements (see Equation (1)).

As illustrated in Figure 1, SkelGIS users write an operation *OP* using the programming interface *PI*. Then, the operation OP is called through an applier *AP*. SkelGIS supports transparently data distribution and communications. This way, a sequential view of the program is provided to the user (Niklaus Wirth's aphorism), while a parallel program is actually written.

SkelGIS implements several DDSs and their related DPMaps, AP and I. In this paper,

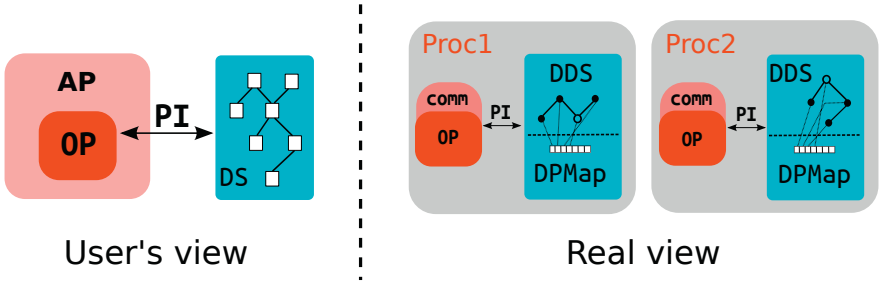


Figure 1: SkelGIS user's view and its actual parallel execution

parallelization of an arterial blood-flow simulation using SkelGIS is studied, thus the rest of this section describes SkelGIS components used to represent networks.

DDS. The DDS which represents a network is called distributed DAG and is denoted *DDAG*. The implementation of the DDAG object is not detailed in this paper, but it uses a modified and parallel version of the "Compressed Sparse Row" storage (CSR) [3]. CSR is a light data structure to store sparse matrices. SkelGIS uses a modified CSR method optimized to store distributed DAGs. This representation is also specifically improved for scientific simulations to optimize accesses to physical border elements and neighborhood elements. the DDAG object manages efficient communications between processors including an overlap of computations with communications. Since a DDAG instantiation is an irregular data structure, its distribution is very important to obtain a good load balancing between processors. This data distribution can be represented as a graph partitioning problem, and is solved in the current version of SkelGIS using a sibling-edges heuristic, adapted to scientific simulations. This data distribution produces sensible results but will be improved in future work using hypergraph partitioning [12] and the partitioner Mondriaan [17].

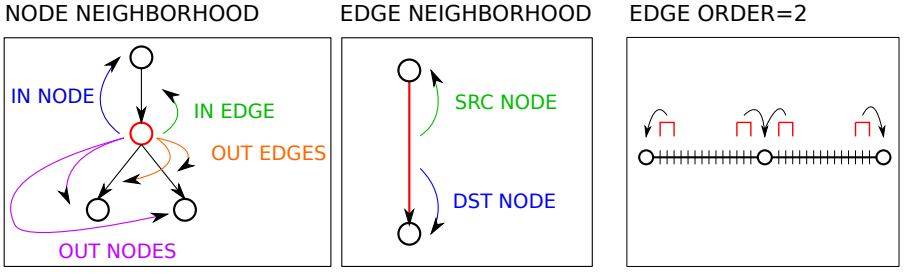
DPMap. An efficient access to data in an irregular structure, as a network, leads to a complex data structure which is heavy to store. Thus, to minimize the memory footprint of the whole data storage, SkelGIS clearly separates the DDAG data structure from data mapped on it. This way, the DDAG is created once and data are stored in lighter objects: DPMaps. Two kinds of DPMaps are available for the DDAG DDS. *DPMap_Nodes* and *DPMap_Edges* to map data respectively on nodes and edges of the network.

AP. A SkelGIS *applier* performs a communication phase, to exchange ghost data between processors, overlapped by a computation phase where each processor applies the operation defined by the user. The prototype of the applier is the following

$$apply\_list : \{DPMap\_Edges\} \times \{DPMap\_Nodes\} \times OP \tag{2}$$

where $\{DPMap\_Edges\}$ and $\{DPMap\_Nodes\}$ denote two sets of $DPMap\_Edges$ and $DPMap\_Nodes$ instantiations used by the operation OP.

PI. An operation OP is a sequential function written by the user. It manipulates DPMap_Edges and DPMap_Nodes instantiations through some programming interfaces. First, three kinds of *iterators* allow the user to navigate through DPmaps. According to Equation (1), there is no computational dependency between elements of the mesh in the same time iteration. Thus, the first kind of iterators moves through nodes or edges and guarantees that the whole DPMap is parsed in an unknown order. Two other iterators are used to parse physical border elements of the network. As already explained, physical border elements of a network are roots and leaves of the DAG. The bracket operator [] is used with an iterator it, $[it]$, to access and update values in DPMaps. Finally, SkelGIS provides some methods to access the neighborhood of an element in the DDAG. Figure 2(a) shows the neighborhood respectively for nodes and edges of the DDAG. This figure illustrates that computations on a node may depend on incoming nodes and edges, and outgoing nodes and edges. Computations on an edge, on the other hand, could be only impacted by its source and destination nodes. As a consequence, DPMap_Edges have two methods which return an iterator to access, respectively, the source and the destination node. In the case of DPMap_Nodes, two methods return a list of iterators to access the incoming and outgoing nodes of the current node, and two methods return the equivalent incoming and outgoing lists for edges. In a distributed simulation, the neighborhood implies some MPI communications between processors. To automatically proceed those communications, before applying an operation, the applier uses the information given by the user when defining the DDS and its DPMaps. In many network scientific simulations, edges represent a part of the space (a section of a river or an artery for example) which is discretized by a one-dimensional mesh as illustrated in Figure 2(b). In such a case, the order of the one-dimensional scheme has to be specified by the user to automatically optimize communications. Thus, for example, in Figure 2(b), only information on two values at the beginning and at the end of the mesh are needed by other processors.



(a) Neighborhood of nodes and edges in a DDAG. (b) 2^{nd} order scheme on edges.

Figure 2: Neighborhood and communications for networks

Thanks to the four SkelGIS concepts dedicated to network simulations, it is possible to totally hide parallelization of codes from the user. Moreover, programming freedom is preserved through a sequential programming style. In the next section, the complex simulation of arterial blood-flow is explained, and the parallelization of this simulation, using SkelGIS, is described.

3 The 1D model of arterial blood-flow

3.1 1D mathematical model

The 1D model of blood-flow in large arteries is usually derived with two main assumptions: axisymmetric velocity profile and large wave length compared with the radius of the vessel, see literature such as [10, 13]. By integrating the Navier Stokes equations across the cross section of the artery, one obtains two PDEs, which link the cross sectional area A, the volumetric flow rate Q and the internal pressure P:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0, \text{ and } \frac{\partial Q}{\partial t} + \frac{\partial}{\partial x}(\frac{Q^2}{A}) + \frac{A}{\rho}\frac{\partial P}{\partial x} = -8\pi\nu\frac{Q}{A}, \tag{3}$$

where x is the longitudinal axis of the artery, t is time and $-8\pi\nu$ is a coefficient of the friction. The first PDE is the conservation law of mass, and the second is the balance of the momentum. The blood density ρ is assumed a constant, ν is the kinematic viscosity (the modeling of shear stress and inertia is discussed in [13]). To close the system we assume that the arterial wall is thin, isotropic, homogeneous, incompressible, and moreover that it deforms axisymmetrically with each circular cross-section independently of the others and it behaves like a Kelvin-Voigt model. We denote the undeformed cross-sectional area by A_0 and the external pressure of the vessel by P_{ext}. Then, the relation linking A and P is:

$$P = P_{ext} + \beta(\sqrt{A} - \sqrt{A_0}) + \nu_s\frac{\partial A}{\partial t}, \tag{4}$$

with the stiffness coefficient β, and the viscosity (of the arterial wall) coefficient ν_s.

3.2 Numerical resolution

A conservation law can be written in the general form

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} = S,$$

where U is the conservation variable, F the flux and S the source term. For finite volume method, the domain is decomposed into finite volumes or cells with vertex x_i as the center of cell $[x_{i-1/2}, x_{i+1/2}]$. In every cell, the conservation law must hold (see [8]),

$$\int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial U}{\partial t}dx + \int_{x_{i-1/2}}^{x_{i+1/2}} \frac{\partial F}{\partial x}dx = \int_{x_{i-1/2}}^{x_{i+1/2}} Sdx.$$

Gauss's theorem is used on the second term and variables are approximated by the averaged values in each cell: $U_i = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} U(x)dx, \quad S_i = \frac{1}{\Delta x}\int_{x_{i-1/2}}^{x_{i+1/2}} S(x)dx$. After the discretization in space, we have the semi-discrete form,

$$\frac{dU_i}{dt} = \Phi(U_{i-2}, ...U_{i+2}) \text{ were } \Phi(U_{i-2}, ...U_{i+2}) = -\frac{(F^*_{i+1/2} - F^*_{i-1/2})}{\Delta x} + S_i.$$

The numerical fluxes $F^*_{i+1/2}$ and $F^*_{i-1/2}$ are given by Rusanov flux (Depending on the approximate approaches on solving the Riemann problem, different numerical fluxes are possible, with slightly different numerical diffusivity. This one is widely used because it is simple and robust, according to [4]). For second order accuracy, a MUSCL (monotonic upwind scheme for

conservation law) linear reconstruction technique is used. This is a scheme with five stencils. The values at x_1 and x_{N+1} are determined by the characteristic method. For the temporal integration, we may apply a 2-step second order Adams-Bashforth (A-B) scheme,

$$\mathbf{U}^{n+1} = \mathbf{U}^n + \Delta t\big(\frac{3}{2}\Phi(\mathbf{U}^n) - \frac{1}{2}\Phi(\mathbf{U}^{n-1})\big).$$

This scheme can be initiated by a forward Euler method.

Arteries are joined at conjunctions: the nodes of the network. We take a simple branching with two daughter arteries as an example. At the node, there are then six boundary conditions, A_p^{n+1} and Q_p^{n+1} for the outlet of the parent artery and $A_{d_1}^{n+1}$, $Q_{d_1}^{n+1}$,$A_{d_2}^{n+1}$ and $Q_{d_2}^{n+1}$ for the inlets of the two daughter arteries. The conservation law of mass and momentum fluxes reads

$$Q_p^{n+1} - Q_{d_1}^{n+1} - Q_{d_2}^{n+1} = 0, \tag{5}$$

$$\frac{1}{2}\rho(\frac{Q_p^{n+1}}{A_p^{n+1}})^2 + P_p^{n+1} - \frac{1}{2}\rho(\frac{Q_{d_i}^{n+1}}{A_{d_i}^{n+1}})^2 - P_{d_i}^{n+1} = 0 \quad i = 1, 2. \tag{6}$$

The pressures P_p^{n+1} and $P_{d_i}^{n+1}$ shall be expressed in cross-sectional area A by the constitutive relation (4). Moreover, the unknowns should match the three outgoing characteristics of the joined arteries [10]. Thus we obtain a nonlinear algebraic system of 6 equations with 6 unknowns, which can be readily solved by Newton-Raphson iterative method with U^n as the initial guess.

3.3 Parallelization of the simulation

This simulation has first been implemented sequentially using C++. Then, two parallel programs have been derived from the sequential code. The first one is an OpenMP version of the sequential code, which applies a coarse-grain parallelization model. This produces a SPMD parallel program (Single Program, Multiple Data) where each processor is in charge of a subpart of the network. Since this program is a shared memory solution, no communications are needed between processors which can directly access values managed by other processors. The advantage of this kind of parallelization is that the sequential code is almost not modified. However, coarse-grain parallelization is not completely implicit as the user has to manage the distribution of the network among processors and specify whether a variable is shared or local. The second parallel version of the simulation has been implemented with SkelGIS. The main function pseudo-code of the simulation is shown in Algorithm 1. This code is very close to the sequential code, but it uses SkelGIS objects and tools instead of homemade data structures. The blood-flow operation pseudo-code is described in Algorithm 2. The operation is organized exactly as the sequential code. Thus, it first deals with roots and leaves of the network to simulate the physical border behavior (lines 1 to 13). Then, as in the sequential code, conjunction nodes of the arterial network (nodes of the DAG) are solved. Finally arteries of the network (edges of the DAG) are solved. For each of these steps, *iterators* are used to move through elements of the network. Most of the sequential code can be kept because an object obtained from a SkelGIS iterator is a plain C++ variable as in the sequential code. Thus, as the SkelGIS program keeps the sequential code structure and most of the sequential code itself, the effort to code the blood-flow simulation with SkelGIS is quite light. In addition to this, the use of SkelGIS data structures avoids the user to implement its own containers. As a result, the SkelGIS code is even simpler than the sequential code.

Algorithm 1: Main function of the blood-flow simulation with SkelGIS

```
1 Instantiation of the distributed network DDAG
2 Instantiation of DPMaps using DDAG
3 Initializations
4 while not end of time iteration do
5 |  applier({DPMap},bloodflow)
6 end
```

Algorithm 2: Sequential SkelGIS bloodflow operation

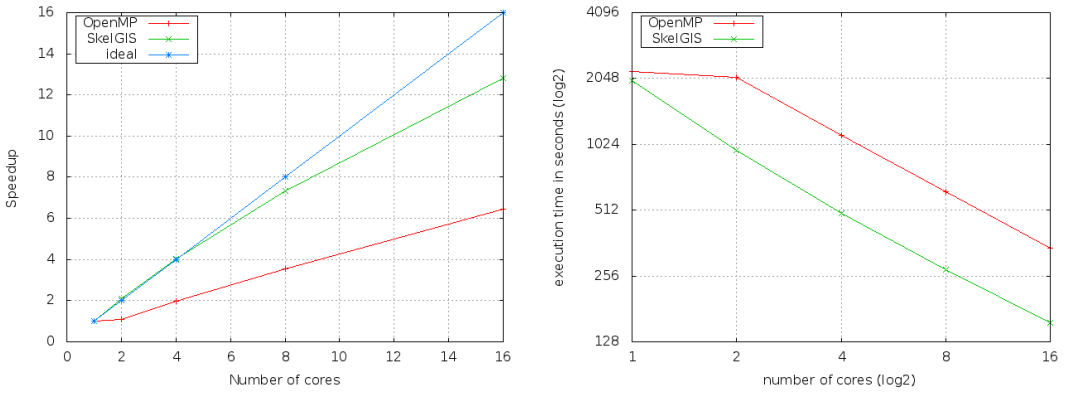
```
   Data: {DPMap}
   Result: Modification of {DPMap}
1  ItR := beginning iterator on roots
2  endItR := end iterator of on roots
3  while ItR≤endItR do
4  |  Use of neighborhood of {DPMap}
5  |  Use of [] on {DPMap}
6  |  ItR++
7  end
8  ItL := beginning iterator on leaves
9  endItL := end iterator of on leaves
10 while ItL≤endItL do
11 |  Use of neighborhood of {DPMap}
12 |  Use of [] on {DPMap}
13 |  ItL++
14 end
15 ItC= beginning iterator on conjunctions
16 endItC := end iterator on conjunctions
17 while ItC≤endItC do
18 |  Use of neighborhood of {DPMap}
19 |  Use of [] on {DPMap}
20 |  ItC++
21 end
22 ItA= beginning iterator on arteries (edges)
23 endItA := end iterator on arteries
24 while ItA≤endItA do
25 |  Use of neighborhood of {DPMap}
26 |  Use of [] on {DPMap}
27 |  ItA++
28 end
```

4 Experiments

This section first presents performance comparisons between OpenMP and SkelGIS parallelizations of the 1D arterial blood-flow simulation. Then, both versions are compared in terms of programming efforts. Finally, as the SkelGIS version is written in MPI and made for distributed memory architectures, it is evaluated on a bigger network with more processors of a cluster. Experiments have been computed on thin/standard nodes of the TGCC-Curie cluster (20th cluster in the top500 list of November 2013). Each of those nodes is equipped with two eight-cores CPU Sandy Bridge clocked at 2.7GHz, 64Go of RAM DDR3 and a local SSD disk. Each experiment has been compiled with the flag -O3, and has been computed 4 times. The average execution time is used in the evaluations of this section.

As explained in the previous section, the OpenMP version uses a coarse-grain parallelization which, most of the time, produces better performance than a fine-grain parallelization but does not totally hide parallelization of codes. The SkelGIS version, on the other hand, produces an MPI program, where parallelization of codes is totally hidden from the user. As the OpenMP version is a shared memory solution, it runs on a single machine. Thus, a single standard node of TGCC-Curie, with 16 cores, has been used for evaluations. An artery network of 4.000 arteries and conjunction nodes has been used for this experiment. Speedups of both versions are presented in Figure 3(a), while Figure 3(b) represents execution times with a logarithmic scale.

First, execution times and speedup of the SkelGIS version are better than the OpenMP



(a) Speedups of OpenMP and SkelGIS versions

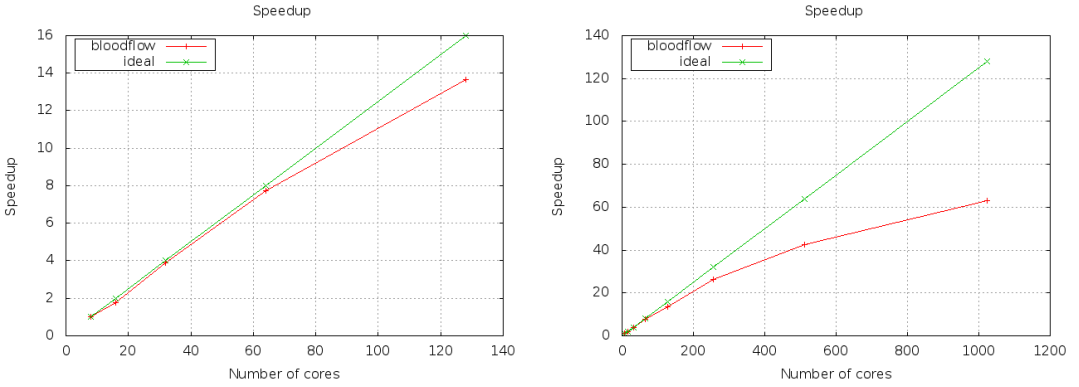
(b) Execution time with logarithmic scale

Figure 3: Experimental comparisons between OpenMP and SkelGIS implementations

version. However, one can note that the difference is exaggerated by the knee at two cores in the scaling graph of the OpenMP version. However, even if this weakness is not taken into account, speedup of the OpenMP version is less steep than the SkelGIS one.

Thus the SkelGIS version of the arterial blood-flow network has interesting performance results compared to an OpenMP equivalent version even if this OpenMP version could probably be improved. Moreover, the SkelGIS version totally hides parallelization of codes while the coarse-grain parallelization of the OpenMP version does not. Thus, it is interesting to evaluate, with metrics, the effort needed to code both parallel simulations. Halstead metrics [11] have been used to estimate the difficulty and the effort to write both source codes. It results that the OpenMP code is 20% more difficult to write than the SkelGIS code, and it requires a programming effort 80% greater than the SkelGIS version. As explained previously, using SkelGIS, the data structure, which represents the network and its connectivity, does not have to be implemented by the user. The user code is thus drastically simplified. As a result, the length and the difficulty of the SkelGIS implementation is more interesting than the OpenMP implementation.

SkelGIS, is a parallel library to create parallel scientific simulations for distributed memory architectures. SkelGIS programs are designed to run on clusters. Thus, the SkelGIS 1D arterial blood-flow simulation has been evaluated on the TGCC-Curie cluster. To show the scalability of the SkelGIS program, a bigger network than the one used in the previous experiment, has been used. It contains 15.000 arteries and 15.000 conjunction nodes. The speedup presented on Figure 4(a) illustrates the speedup from 1 to 128 cores and the speedup presented on Figure 4(b) represents evaluations from 1 to 1024 cores. From 1 to 128 cores, the speedup is almost ideal and linear, which is a very good result for network simulations, especially for an implicit parallelism solution where no parallel code has been written by the user. With more cores, the speedup begins to fall. This is probably due to a weakness in the data distribution implemented. This implementation should be improved to avoid load imbalance and to decrease the communication cost. Two improved data distributions are under progress.



(a) SkelGIS speedup from 8 to 128 cores

(b) SkelGIS speedup from 8 to 1024 cores

Figure 4: Speedups of SkelGIS implementation on a network of 15000 arteries

5 Conclusion

In this paper has been presented the implicit parallelism library SkelGIS for the specific case of network simulations. Indeed, SkelGIS can be used for networks which can be represented as directed acyclic graphs, and the four concepts of SkelGIS have been explained for this case. A real case-study: the blood-flow simulation in an arterial network, has been described, and numerical schemes of both conjunction nodes and arteries have been explained. Three experiments illustrate that, first, the obtained SkelGIS code is efficient on a single multi-core machine compared to an equivalent OpenMP program. Second, the effort needed to implement the simulation is greater with OpenMP than with SkelGIS. Finally, the SkelGIS simulation has a very good scalability on clusters. As explained previously, SkelGIS has already been implemented for two-dimensional regular meshes. A work in progress is to propose a SkelGIS implementation for general networks, thus the work presented in this paper on DAGs will be generalized to graphs. After this, SkelGIS will be implemented for unstructured meshes and later for adaptive meshes to be able to solve most scientific simulations in parallel.

References

[1] Karsten Ahnert and Mario Mulansky. Odeint - solving ordinary differential equations in C++. *CoRR*, abs/1110.3397, 2011.

[2] S. Balay, W. D. Gropp, L. Curfman McInnes, and B. F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[3] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, Philadelphia, PA, 1994.

[4] F. Bouchut. *Nonlinear stability of finite volume methods for hyperbolic conservation laws and well-balanced schemes for sources*. Birkhäuser, 2004.

[5] A. Buss, Harshvardhan, I. Papadopoulos, O. Pearce, T. Smith, G. Tanase, N. Thomas, X. Xu, M. Bianco, N. M. Amato, and L. Rauchwerger. Stapl: standard template adaptive parallel library.

In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 14:1–14:10. ACM, 2010.

[6] Hélène Coullon, Minh-Hoang Le, and Sébastien Limet. Parallelization of shallow-water equations with the algorithmic skeleton library skelgis. In *ICCS*, volume 18 of *Procedia Computer Science*, pages 591–600. Elsevier, 2013.

[7] Hélène Coullon and Sébastien Limet. Algorithmic skeleton library for scientific simulations: Skelgis. In *HPCS*, pages 429–436. IEEE, 2013.

[8] O. Delestre and P.-Y. Lagrée. A well balanced finite volume scheme for blood flow simulation. *International Journal for Numerical Methods in Fluids*, page doi: 10.1002/fld.3736, 2012.

[9] Z. DeVito, N. Joubert, F. Palacios, S. Oakley, M. Medina, M. Barrientos, E. Elsen, F. Ham, A. Aiken, K. Duraisamy, E. Darve, J. Alonso, and P. Hanrahan. Liszt: A domain specific language for building portable mesh-based pde solvers. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 1–12. ACM, 2011.

[10] L. Formaggia, A. Quarteroni, and A. Veneziani. *Cardiovascular Mathematics: Modeling and simulation of the circulatory system*, volume 1. Springer, 2009.

[11] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science, New York, USA, 1977.

[12] Bruce Hendrickson and Tamara G. Kolda. Graph partitioning models for parallel computing. *Parallel Comput.*, 26(12):1519–1534, November 2000.

[13] P.-Y. Lagrée. An inverse technique to deduce the elasticity of a large artery. *EPJ Applied Physics*, 9(2):153–164, 2000.

[14] Brigitte Lucquin and Olivier Pironneau. *Introduction to Scientific Computing*. John Wiley & Sons, Inc., New York, NY, USA, 1997.

[15] W.E. Milne. *Numerical Solution of Differential Equations*. Applied Mathematics Series. John Wiley and Sons, 1953.

[16] GR Mudalige, MB Giles, I. Reguly, C. Bertolli, and PHJ Kelly. Op2: An active library framework for solving unstructured mesh-based applications on multi-core and many-core architectures. In *Innovative Parallel Computing (InPar)*, pages 1–12. IEEE, 2012.

[17] Brendan Vastenhouw and Rob H. Bisseling. A two-dimensional data distribution method for parallel sparse matrix-vector multiplication. *SIAM Review*, 47(1):67–95, 2005.

[18] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1978.