Blood flow arterial network simulation with the implicit parallelism library SkelGIS

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

[1] LIFO - Université d'Orléans, Bat IIIA, rue Leonard de Vinci, 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

Abstract

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 [9, 10]. Three numerical methods are more popular. The first one, based on the derivation from Taylors polynomial, is called *finite difference method*. In this 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 well known resolution method is called *finite element method*. The exact solution is approximated by a continuous and piecewise polynomial function. The domain is often discretized by a triangulation. Numerical methods discretize time and space. When a discretization in space is done, a *mesh* of the space is created. In a program, the mesh is represented by a data structure which models the connectivity between elements. Moreover, in a program, discretization of time is translated to a main loop for time iterations. In each time iteration, the scheme obtained by the reosolution method is computed. The type of scientific simulations we are interested in can be represented by the 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 $U_t(x)$. This neighborhood is often called *stencil*. In other words, quantities of the element x at a time iteration t is a function of quantities for this element and its neighborhood at time iteration $t-1$. This representation is a simplification of the set of PDEs (3b) in Section 3). The neighborhood needed by a simulation is linked to the order of the resolution method. Precision of a simulation is defined by its order, and the order gives information on needed neighborhood. The last important point of sicentific simulations is the concept of physical border of the domain. In a real-world process, the domain is not limited, however, in a simulation 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. Similarly, access to super-computers is becoming easier with a growing number of clusters over the world. As a result, it becomes crucial for scientists of all domains to write parallel programs. For this reason, and because of lack of time and ressources 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. Actually, if the solution is too much generic, it works for most scientific problems, but it could be difficult to use. On the other hand, if a solution is too much 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 that can be represented by the scheme (1). However, this class of simulations is also very large as most scientific simulations are solved this way. The implicit parallelism library ScaLAPACK [4] (Scalable Linear Algebra PACKage) that embed PBLAS [11] (Parallel Basic Linear Algebra Subprograms) is very well known and widely used. This library is specifically made to solve linear algebra problems in parallel. Some libraries are closer to SkelGIS as odeint [1] or PETSc [2]. Odeint is an implicit parallelism library to solve ordinary differential equations. This library proposes implicit parallelism on GPUs with CUDA. PETSc solves partial differential equations and proposes implicit parallelism on GPUs, with CUDA, CPUs, with MPI and pthreads programming, and hybrid systems. The main difference with SkelGIS is the view that the user get on the simulation and the programming freedom of the user. In SkelGIS, the user does not call specific tools to solve differential equations, as for example *interpolate* or *jacobian* functions etc. SkelGIS proposes a new kind of implicit parallelism solution to solve partial differential equations where the user codes its own sequential 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 could 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 five 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 obtained programs were convincing compared to equivalent MPI codes and compared to the coding effort needed. SkelGIS aims to apply its five concepts to different kind of simulations and to different kind of meshes. This paper deals with the case of network simulations which highlights different kind of difficulties than a simulation on a two-dimensional regular mesh. In a network simulation, the domain is divided in two different kind of elements with a 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 polluant 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. In this paper is studied the use of SkelGIS for network simulations. As far as we know, no specific implicit parallelism solutions exist to solve network simulations. However, using PETSc, sparse matrices can be defined and partial differential equations can be solved on it. As a network can be translated to a sparse matrix, it is possible to write network simulations using PETSc. However, two different kind of meshes have to be applied, then multiple kind of sparse matrices, for nodes and edges, have to be managed by the user. 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 with SkelGIS. The paper 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. Performances results are compared to an OpenMP version of the same simulation, and efficiency on a big network is evaluated. Finally, conclusion and perspectives on this work are given.

2 SkelGIS implicit parallelism library

SkelGIS library is an implicit parallelism solution for scientific simulations. It aims to offer a transparent access to parallel computing. Niklaus Wirth's aphorism [12] "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 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. First, a *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. Second, a *DPMap* is a data mapping which represents data of the simulation (quantities it uses) and its mapping on the DDS. Third, an AP is an applier. An applier 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, I represents the programming interface of SkelGIS. I is used by programmers 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 (as needed in the equation (1)).

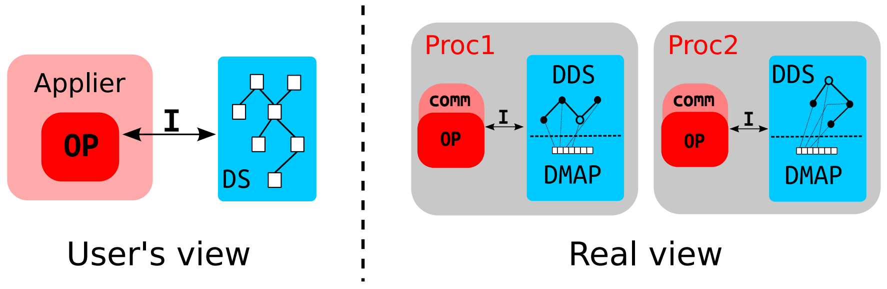


Figure 1: SkelGIS user's view and its actual parallel execution

As illustrated in Figure 1, SkelGIS users write an operation OP using the programming interface I. 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 DPMap, AP and I. In this paper, 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. DDAG manages efficient communications between processors including communications/computations overlapping. Since DDAGs are irregular data structures, their distribution is very important to obtain a good load balancing between processors. This is done using a sibling-graph model adapted to scientific simumlations.

DPMap. To obtain an efficient access to data in an irregular structure as a network, the data structure has to be complex and 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. Therefore, the DDAG is create 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.

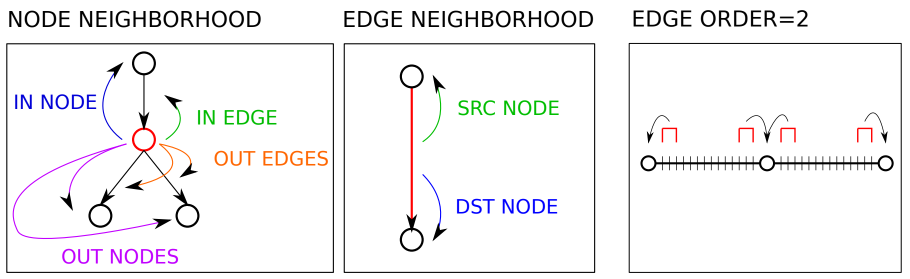
AP. A SkelGIS *applier* performs a communication phase to exchange ghost data between processors followed by a computation phase where each processor applies the operation defined by the user. The prototype of the applier is the following

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

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

I. The operation OP is a sequential function written by the user. It manipulates DPMap_Edges and DPMap_Nodes instanciations through some programming interfaces. First, three kinds of *iterators* allows the user to navigate through DPmaps. According to Equation (1), there is no computional dependency between elements of the mesh of the same time iteration. Thus, the first kind of iterator moves through nodes or edges. Iterators guarantee that the whole DPMap is parsed but the order is unknown. 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 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.

First, computation on a node could be impacted by incoming nodes and edges, and outgoing nodes and edges. Secondly, computation on an edge could be impacted by its source and destination nodes. As a consequence, DPMap_Edges has two methods to obtain an iterator that gives access to the source and destination nodes, respectively. Neighborhood for the cas of DPMap_Nodes is more complicated. Two methods return a list of iterators to give access to the incoming and outgoing nodes of the current node. And two methods return the equivalent incoming and outgoing lists for edges. In a simulation, the needed 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 scientific simulations, edges of a network represent a part of the space (a section of a river or an artery for example) that is discretized by a one-dimensional mesh as in Figure 2(b). In such a case, the order of the one-dimensional scheme



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

Figure 2: Neighborhood and communications for networks

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 end and at the begining of the mesh could be needed by other processors.

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 details of the derivation of the 1D model can be found in literature, such as [?, ?, ?, ?]. We stress the two main assumptions usually held in most applications: axisymmetric velocity profile and large wave length compared with the radius of the vessel. Then, integrating the Navier Stokes across the section $A(x,t)$ of the artery, the longitudinal axis of the artery being x (the time is t), one obtains the 1D arterial blood flow model linking $Q(x,t)$ the volumetric flow rate and the section $A(x,t)$. First the conservation of mass, and second, the momentum equation:

$$\frac{\partial A}{\partial t} + \frac{\partial Q}{\partial x} = 0, \tag{3a}$$

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

where P is the internal pressure. The blood density ρ is assumed a constant, ν is the kinematic viscosity (the modelling of shear stress is discussed in [?, ?] see some other details of modelling inertia in [?]). To close the system we suppose a Kelvin-Voigt model for simplicity[?, ?]. 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. 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, We note that in absence of the wall viscosity we retrieve the classical Hooke's law.

3.2 Numerical resolution

3.2.1 Descretization of governing equations of each artery

In this section, we mainly follow the presentation [**?**] but with a different temporal integration method. For finite volume method, the domain is decomposed into finite volumes or cells with vertex x_i as the center of cell $[x_{i-1/2}, x_{i+1/2}]$, see Figure **??**. In every cell, the conservation law must hold,

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

Application of Gauss's theorem on the second term and using average values:

$$U_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} U(x) dx, \quad S_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} S(x) dx.$$

Thus Eq. 3.2.1 turns into an ordinary differential equation

$$\frac{dU_i}{dt} = -\frac{(F|_{x_{i+1/2}} - F|_{x_{i-1/2}})}{\Delta x} + S_i. \tag{5}$$

We have a local Riemann problem at each interface of neighboring cells, since $U_{i+1/2-}$ and $U_{i+1/2+}$, the left limit of U_i and the right limit of U_{i+1} at $x_{i+1/2}$ respectively, are not equal in general. By solving the Riemann problem, a numerical flux F^* can be obtained. Depending on the approximate approaches on solving the Riemann problem, different numerical fluxes are possible (with more or less numerical diffusivity), such as HLL flux [**?**, **?**]. Among them, Rusanov (or called local Lax-Friedrichs) flux is widely used, it is simple and robust, according to reference [**?**]. If $\mathbf{U}_-$ and $\mathbf{U}_+$ are equal to the average values at the cells, the scheme will be of first order accuracy. Reconstructions of $\mathbf{U}_-$ and $\mathbf{U}_+$ from $\mathbf{U}$ are necessary for a scheme of higher resolution. Here a MUSCL (monotonic upwind scheme for conservation law) is one popular slope limited linear reconstruction technique is used.

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 with the reconstructed values at the two sides of the interfaces. Note that this is a scheme with five stencils. The values at x_1 and x_{N+1} are determined by the aforementioned characteristic method. One ghost cell at each end of the computational domain is needed and we approximate the values at these cells by the ones at the neighboring boundary cells. 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.

3.2.2 Treatment of conjunctions

The arteries are segments joined at nodes: a parent vessel with two daughter arteries. At the branching point, 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. From the physical point of view, we have to preserve the conservation of mass flux

$$Q_p^{n+1} - Q_{d_1}^{n+1} - Q_{d_2}^{n+1} = 0, \tag{6a}$$

and conservation of momentum flux

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

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 outgoing characteristics of the joined arteries should be matched. In the parent artery, $(W_1)_p^{n+1}$ is given by the data on the n-th time step with the interpolation formula (**??**). It must be equal to $W_1(U_p^{n+1})$ which is given by relation (**??**). Then

$$(W_1)_p^{n+1} - W_1(U_p^{n+1}) = 0. \tag{6c}$$

The same principle holds for W_2 on the two daughter arteries,

$$(W_2)_{d_i}^{n+1} - W_2(U_{d_i}^{n+1}) = 0 \quad i = 1, 2. \tag{6d}$$

Combining Eqs. (6a), (6b), (6c) and (6d), there are 6 Eqs. with 6 unknowns. This nonlinear algebraic system can be readily solved by Newton-Raphson iterative method with U^n as the initial guess. Usually a very few iterations are enough for a satisfactory accuracy.

3.3 Parallelization of the simulation

This simulation has been first implemented by a sequential program. Then, two parallel programs have been derived from the sequential code. The first one is an OpenMP version of the sequential code using a coarse-grain parallelization model. This produces a SPMD parallel program (Single Program, Multiple Data) where each processor is in charge of a sub-part of the network. Since this program is a shared memory solution, no comunications are needed between processors that 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 transparent 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 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 bloodflow operation is described in Algorithm 2. The operation is organized exacly 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, conjonction 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 the objects obtained from an iterator are plain C++ variables 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 bloodflow simulation with SkelGIS is quite light. In addition to this, the use of SkelGIS data structures avoids the user to implement its own data structures. As a result, the SkelGIS code is even simpler than the sequential code.

Algorithm 1: Main function of the bloodflow 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

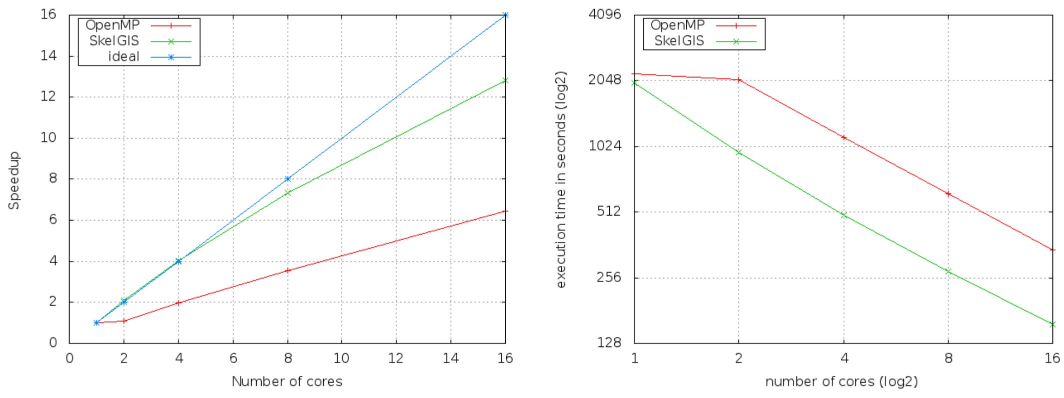
```
   Data: {DPMap}
   Result: Modification of {DPMap}
1  ItR := beginning iterator on root nodes
2  endItR := end iterator of on root nodes
3  while $ItR \leq endItR$ do
4      Use of neighborhood of $\{DPMap\}$
5      Use of set and get of $\{DPMap\}$
6      ItR++
7  end
8  ItL := beginning iterator on leaf nodes
9  endItL := end iterator of on leaf nodes
10 while $ItL \leq endItL$ do
11     Use of neighborhood of $\{DPMap\}$
12     Use of set and get of $\{DPMap\}$
13     ItL++
14 end
15 ItC= beginning iterator on conjonctions
16 endItC := end iterator on conjonctions
17 while $ItC \leq endItC$ do
18     Use of neighborhood of $\{DPMap\}$
19     Use of set and get of $\{DPMap\}$
20     ItC++
21 end
22 ItA= beginning iterator on arteries (edges)
23 endItA := end iterator on arteries
24 while $ItA \leq endItA$ do
25     Use of neighborhood of $\{DPMap\}$
26     Use of set and get of $\{DPMap\}$
27     ItA++
28 end
```

4 Experiments

This section first presents performances comparisons between the OpenMP and the SkelGIS parallelization of the 1D arterial bloodflow simulation. Then, both versions are compared in terms of programming efforts. Finally, as the SkelGIS version is written in MPI and made for clusters computations, it is evaluated on a bigger network with more processors of a cluster. Experiments have been computed on the TGCC-Curie cluster. More precisely on thin/standard nodes of the cluster. 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 computed 4 times and the average execution time has been used for results presented in this section.

As explained in the previous section, the OpenMP version uses a coarse-grain parallelization which, most of the time, produces better parformances than a fine-grain parallelization but does not totally hide parallelization of codes. The SkelGIS version, on the other hand, produces an MPI program, but parallelization of codes is totally hidden from the user through four concepts. As the OpenMP version is a shared memory solution, it can only be run on a single machine. Thus, a single standard node of TGCC-Curie, with 16 cores, has been used for evaluations. The human artery network has been used for this experiment. This network is very light as it contains only 55 arteries and conjunction nodes. Speedups of both version 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

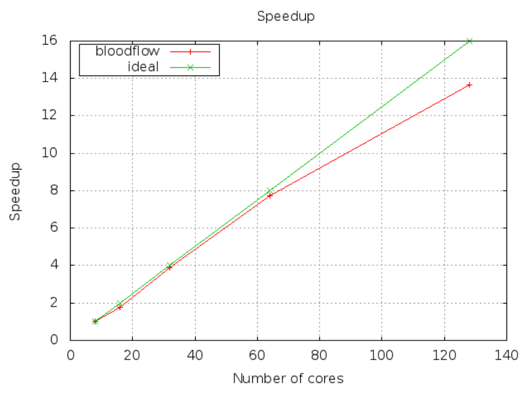
version. However, one can note that the difference is exaggerated by the bad speedup from one to two cores in the OpenMP version. However, even if this weakness is not taken into account, speedup of the OpenMP version is less steep than the SkelGIS version.

SkelGIS version of the arterial bloodflow network have interesting performances results compared to an OpenMP equivalent version even if this OpenMP version could 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. We used Halstead [8] metrics 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.

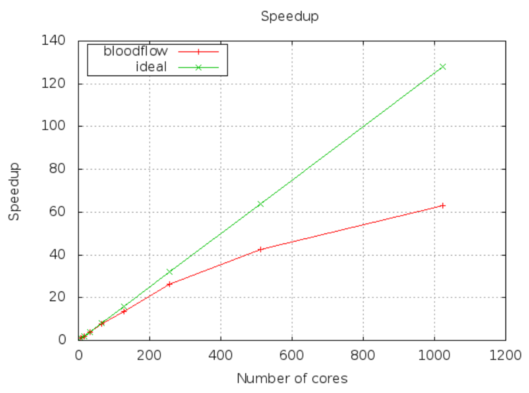
SkelGIS, is a parallel library to create parallel scientific simulations for distributed memory architectures. SkelGIS programs are made to run on clusters. Thus, the SkelGIS 1D arterial bloodflow simulation has been evaluated on the TGCC-Curie cluster. To obtain an interesting evaluation, the size of the arterial network is bigger than the one used in the previous experiment. Thus, a network of 15.000 arteries and 15.000 conjunction nodes have been used. Actually, this network is the network of the Loire river. The speedup presented on Figure 4(a) illustrates results from one to 128 cores. And the speedup presented on Figure 4(b) represents evaluations from one to 1024 cores. From one 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 result could be improved working on a better network distribution and adding some optimizations in SkelGIS.

5 Conclusion

In this paper has been presented the implicit parallelism library SkelGIS for the specific case of network simulations. Actually, SkelGIS can be used for networks which can be represented as directed acyclic graphs, and the four concepts of SkelGIS have been explained in this case.



(a) SkelGIS speedup from 8 to 128 cores



(b) SkelGIS speedup from 8 to 1024 cores

Figure 4: Speedups of SkelGIS implementation on the Loire network

The real-case simulation of the blood flow in the 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 OpenMP equivalent program, second, the effort needed to implement the simulation with OpenMP is greater than using SkelGIS, and finally, the SkelGIS simulation has very good results 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 adaptative meshes.

References

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

[2] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *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] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[5] Antal Buss, Harshvardhan, Ioannis Papadopoulos, Olga Pearce, Timmie Smith, Gabriel Tanase, Nathan Thomas, Xiabing Xu, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger. Stapl: standard template adaptive parallel library. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*, SYSTOR '10, pages 14:1–14:10, New York, NY, USA, 2010. ACM.

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

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

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

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

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

[11] Ravi Reddy, Alexey Lastovetsky, and Pedro Alonso. Heterogeneous pblas: A set of parallel basic linear algebra subprograms for heterogeneous computational clusters. Technical report, 2008.

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