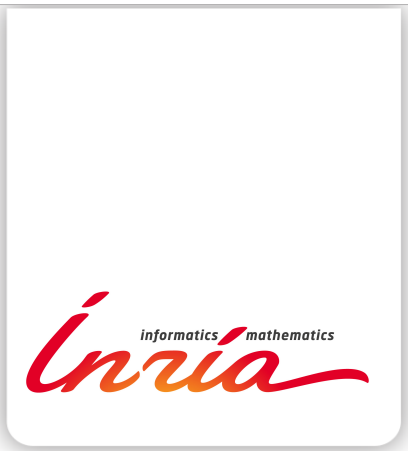
Component-based implicit parallelism model for multi-stencil numerical simulations

Héléne Coullon, Christian Perez

RESEARCH REPORT
N° 7003
April 2015
Project-Teams Avalon

ISSN 0249-6399 ISRN INRIA/RR--7003--FR+ENG



Component-based implicit parallelism model for multi-stencil numerical simulations

Héléne Coullon*†, Christian Perez‡

Project-Teams Avalon

Research Report n° 7003 — April 2015 — 6 pages

Abstract: abstract

Key-words: No keywords

* Footnote for first author
† Shared foot note
‡ Footnote for second author

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Exemple de document utilisant le style rapport de recherche Inria

Résumé : Resume en francais

Mots-clés : mots-cles

Contents

1 Introduction

2 Multi-stencil programs

To numerically solve a set of PDEs, iterative methods are frequently used to approximate the solution by a step by step phenomena. Thus, the continuous time and space domains are discretized so that a set of numerical computations are iteratively (time discretization) applied on a mesh (space discretization). In other words, the PDEs are transformed to a set of numerical computations applied at each time step on all elements of the discretized space domain. Among the numerical computations is found a set of numerical schemes, also called *stencil computations*. A formal definition of a *stencil program*, and the parallelization of such program, are presented in this section.

2.1 Definitions

A mesh $\mathcal{M}$ defines the discretization of the continuous space domain Ω of a set of PDEs.

Definition 1 *A mesh is a connected undirected graph without bridges.*

E_i denotes a set of elements constructed by a function $elem_i$ which defines a precise association of nodes and edges of a mesh. If we denote *nodes* the function to get nodes of a mesh and *edges* the same for edges, the funtion $elem_i$ is defined as

$$elem_i : nodes(\mathcal{M}) \times edges(\mathcal{M}) \rightarrow E_i$$

For example, the set of cells E_0 in a Cartesian 2D mesh could be defined by exactly four nodes and four edges connected as a cycle. But we could also define another set of elements E_1 as the simple set of nodes etc.

A mesh can be structured (as Cartesian or curvilinear meshes), unstructured, regular or irregular (without the same topology for each element) and hybrid if mixing those.

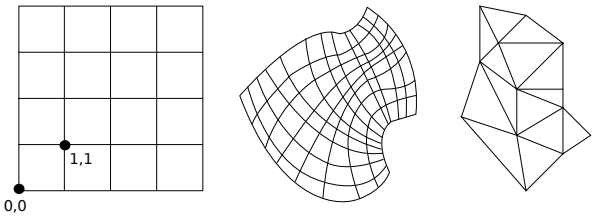


Figure 1: From left to right, Cartesian, curvilinear and unstructured meshes.

The discretization of the continuous time domain $\mathcal{T}$ is denoted T. T is responsible for the iteration time steps of the numerical simulation.

In a numerical simulation a set of data, or quantities, are applied onto the mesh and represent the set of values to compute, or to use, for computation. We denote the set of data applied on the mesh by Δ, such that $\delta \in \Delta$ is a function which associates each element $el \in E_i$ to a value $v \in V$,

$$\delta : E_i \to V$$

One can notice that in applied mathematics, the signature of δ would be $\delta : E_i \times T \to V$, however when programming a numerical simulation it is not wise to store all values of each time iteration.

Finally, in a numerical simulation are performed a set of ordered numerical computations denoted Γ. A numerical computation in Γ is denoted

$$c(R, w, D, e).$$

In this definition $R \subset \Delta, w \in \Delta$ are respectively the data read and written to compute the numerical expression e (a single data is written in a single computation). D is one of the subsets $E_i \subset \mathcal{M}$. It has to be noticed that at each time iteration, all the elements of a mesh are computed. However, it happens that the computation of the mesh elements is splitted in different computations (for example the computation of the physical border). In this case additional E_i can be specified for the mesh $\mathcal{M}$. The numerical expression e computes the data w from a set of data R.

$$e : R \to w$$

A *multi-stencil program* is defined by the quadruplet

$$\mathcal{MSP}(T, \mathcal{M}, \Delta, \Gamma), \tag{1}$$

If the number of computations in Γ is $card(\Gamma) = m$, such that $\bigcup_{i=0}^{m-1} c_i = \Gamma$, then $\bigcup_{i=0}^{m-1} R_i \cup w_i \subseteq \Delta$.

A computation $c \in \Gamma$ can be of two different types. The first type is called a *stencil computation* and involves an additional information denoted $\mathcal{N}$ which represents a neighborhood function around an element

$$\mathcal{N} : E_i \to E_k \times E_k \times ... \tag{2}$$

, where E_k is a set of elements of $\mathcal{M}$ which could be equal to E_i or not. In other words, from an element $el \in E_i$ the function $\mathcal{N}$ returns a set of elements of E_k which represents the

neighborhood of *el*. The function $\mathcal{N}$ is also sometimes called the *stencil shape*, or the *stencil* in applied mathematics.

A *stencil computation* is a quintuplet

$$s(R, w, D, e, \mathcal{N}), \tag{3}$$

where $R \subset \Delta, w \in \Delta$ are read and written in the numerical expression e, and D is one the subsets E_i. In a stencil computation s, $\forall d \in D$, the stencil numerical expression e is applied such that $w(d) = e(R(d), R(\mathcal{N}(d))$. In this work, a stencil computation $s(R, w, D, e, \mathcal{N})$ always verifies $R \cap w = \emptyset$, otherwize an implicit numerical scheme has to be solve which is over the scope of this paper.

Figure **??** gives an example of a stencil computation $s(R, w, D, e, \mathcal{N})$, on a two dimensional Cartesian mesh, where $R = \{A, B\}$, $w = C$, and where for $(x, y) \in D$

$$e(R(x, y), R(\mathcal{N}(x, y)) = A(x, y) + B(x, y + 1) + B(x, y - 1) + B(x + 1, y) + B(x - 1, y).$$

Finally, the second type of numerical computation is a local computation where e does not involve a neighborhood function $\mathcal{N}$

$$l(R, w, D, e) \tag{4}$$

2.2 Parallelization

Multi-stencil mesh-based numerical simulation can be parallelized in various ways and is an interesting kind of application to take advantage of modern heterogeneous HPC architectures, mixing clusters, multi-cores CPUs, vectorization units, GPGPU and many-core accelerators.

The *SPMD* (Single Program, Multiple Data) parallelization technique which is called *coarse-grain data parallelism* in the rest of this paper, is broadly used because of its scalability. This parallelization concept is the well-known domain decomposition technique where the mesh is splitted in balanced sub-meshes for available resources, and the same program is applied on each sub-part of the initial mesh. Because of the good locality degree of numerical simulations, only a small number of additional elements, called a *recorvery* or a *ghost* area, are needed to get information computed on another resource. This ghost area is however unavoidable to correctly compute stencils because of the neighborhood $\mathcal{N}$. As the amount of data to synchronize between resources is rather small compared to the amount of computation (small neighborhood), and also by the addition of recovery of synchronizations by computations, this parallelization technique has proved many times its efficiency and scalability on numerical simulations []. This parallelization technique can be applied on shared and distributed memory architectures (clusters, multi-core CPUs), and it is one of the most efficient parallelization technique if the numerical simulation has to be executed on a distributed memory architecture such as a cluster. Figure **??** illustrates this parallelization technique.

More formally, in a SPMD parallelization of a multi-stencil program $\mathcal{MSP}(T, \mathcal{M}, \Delta, \Gamma)$ consists in, first, a partitioning of the mesh $\mathcal{M}$ in n balanced sub-meshes (for n resources) $\{\mathcal{M}_0, \ldots, \mathcal{M}_{n-1}\}$. This step can be performed by an external graph partitionner [] and is not adressed by this paper. The second step of the parallelization is to identify in the ordered list of computation Γ when synchronizations are needed between resources.

Definition 2 *A synchronisation is needed between two computations c_1 and c_2, denoted $c_1 \ll c_2$, if $c_2 = s_2(R_2, w_2, D_2, e_2)$, and $w_1 \cap R_2 \neq \emptyset$.*

Actually, a synchronization can only be needed by the data read in a stencil computation, and only if this data has been modified before, which means that it has been written before. A new ordered list of computations Γ' is obtained from the insertion of the synchronizations at the good places. The final step of this parallelization is to run the same ordered list of computation Γ' on each resource. For each resource $0 \leq k \leq n-1$ a multi-stencil program defined by

$$\mathcal{MSP}_k(T, \mathcal{M}_k, \Delta_k, \Gamma'), \tag{5}$$

is runned.

PARALLEL,SYNC definitions + algorithm

3 Component Stencil Model

3.1 A primitive component model

Definitions of a primitive component (ports, interfaces), a component instanciation, an assembly + simple semantics on the behavior of a primitive component. Cite L2C as an implementation of such a model + introduction of notations (use,provide,use-multiple,sync).

3.2 Control components

Definition of three control components, and of the kernel component. Explain the behavior.

3.3 Transformation to a parallel assembly

Transformation from a parallel algorithm using the directives: PARALLEL, PARALLEL SECTIONS, SECTION and FORALL, to a component assembly

4 Component Stencil Language

4.1 CSL language and example

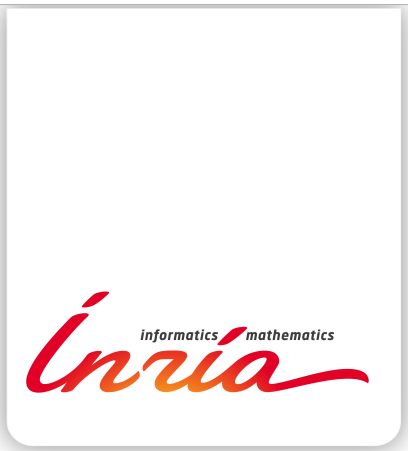
4.2 CCSL compiler and example

4.3 Evaluation

5 Related work

stencil compilation: Pochoir, PATUS etc.
stencil programs: Lizst, OP2
control in components: X-MAN (The New Component Model), STCM, Kell-calculus

6 Conclusion



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399