Component-based parallelization of multi-stencil programs

Héléne Coullon, Christian Perez

RESEARCH REPORT
N° 7003
April 2015
Project-Teams Avalon

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

Component-based parallelization of multi-stencil programs

Héléne Coullon[*][†], Christian Perez[‡]

Project-Teams Avalon

Research Report n° 7003 — April 2015 — 9 pages

Abstract: abstract

Key-words: No keywords

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

Exemple de document utilisant le style rapport de recherche Inria

Résumé : Resume en francais

Mots-clés : mots-cles

Contents

1 Introduction

2 Computational model of 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 and is defined as followed.

Definition *A mesh is a connected undirected graph $\mathcal{M} = (V, E)$, where V is the set of vertices and E the set of edges. The set of edges E of a mesh $\mathcal{M} = (V, E)$ does not contain bridges.*

Definition D_i is a set of elements of a mesh $\mathcal{M} = (V, E)$, constructed by a function $domain_i$ which defines a precise association between V and E, $domain_i : V \times E \rightarrow D_i$.

For example, the set of cells D_0 in a Cartesian 2D mesh could be defined by exactly four vertices and four edges connected as a cycle. But we could also define another set of elements D_1 as the simple set of vertices V etc.

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

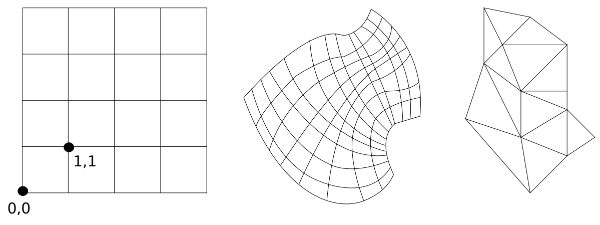

Figure 1: From left to right, Cartesian, curvilinear and unstructured meshes.

Definition The discretization of the continuous time domain $\mathcal{T}$ is denoted T such that $\forall t_i, t_{i+1} \in T, \exists \Delta t \in \mathbb{R}$, $t_{i+1} = t_i + \Delta t$. Thus, 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.

Definition The set of data applied on the mesh is denoted by Δ, such that $\delta \in \Delta$ is a function which associates each element $d \in D_i$ to a value $v \in V$, $\delta : D_i \to V$.

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.

Definition A numerical expression exp is a function which represents how to compute, for an element $d \in D_i$, a data $w \in \Delta$ (written data) with a set $R \subset \Delta$ of input data (read data), $\exp : R \times D_i \to w \times D_i$.

Definition A computation c of a numerical simulation is defined as $c(R, w, D, \exp)$, where $R \subset \Delta, w \in \Delta$ and exp a numerical expression $\exp : R \to w$. D is one of the subsets $D_i \subset \mathcal{M}$, such that $w : D \to V$.

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 D_i can be specified for the mesh $\mathcal{M}$.

Definition The set of n ordered computations of a numerical simulation is denoted $\Gamma = [c_i]_{0 \leq i \leq n-1}$, such that $\forall c_i, c_j$ with $i \leq j$, c_i is computed before c_j, and c_j can be computed only when c_i is finished.

Definition Finally, a *multi-stencil program* is defined by the quadruplet $\mathcal{MSP}(T, \mathcal{M}, \Delta, \Gamma)$.

Definition The neighborhood $\mathcal{N}$ of an element $d \in D_i$ is a function to define a set of elements in any $D_k \subset \mathcal{M}$, $\mathcal{N} : D_i \to D_k \times D_k \times \ldots$.

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

A computation $c \in \Gamma$ can be of two different types. The first type is called a *stencil computation.*

Definition A *stencil computation* is defined as a quintuplet $s(R, w, D, \exp, \mathcal{N})$, where $R \subset \Delta, w \in \Delta$, D is one the subsets D_i and $w : D \to V$.

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

Figure 2 gives an example of a stencil computation $s(R, w, D, \exp, \mathcal{N})$, where $\mathcal{M}(V, E)$ is a two dimensional Cartesian mesh. A single domain D is defined in this example and is composed of cells formed by a cycle of four vertices $v \in V$ and four edges $e \in E$. Furthermore, in this example $R = \{A\}$, $w = B$, and for $(x, y) \in D$ the neighborhood function is

$$\mathcal{N} : (x, y) \to \{(x, y + 1), (x, y - 1), (x + 1, y), (x - 1, y)\}.$$

Finally, the numerical expression of this example is

$$\exp(A(x, y), A(\mathcal{N}(x, y)) = B(x, y) = A(x, y)+(A(x, y+1)+A(x, y-1)+A(x+1, y)+A(x-1, y))/4.$$

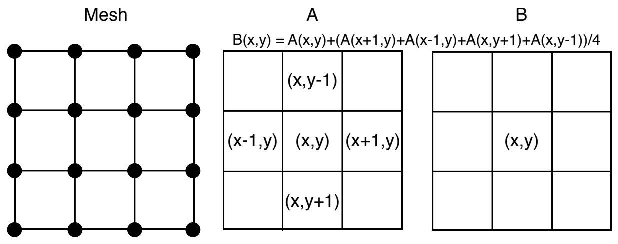

Figure 2: Example of a stencil computation.

Finally, the second type of numerical computation is a local computation.

Definition A local computation is a quadruplet $l(R, w, D, \exp)$, where e does not involve a neighborhood function $\mathcal{N}$.

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.

Coarse-grain data parallelism. In a data parallelization technique, the idea is to split the data on which the program is computed in balanced sub-parts, one for each available resource. The same sequential program can afterwards be applied on each sub-part simultaneously, with some additioinal synchronizations between resources to update the data not computed locally, and thus to guarantee a correct result.

More formally, the data parallelization of a multi-stencil program $\mathcal{MSP}(T, \mathcal{M}, \Delta, \Gamma)$ consists in, first, a partitioning of the mesh $\mathcal{M}$ in p balanced sub-meshes (for p resources) $\{\mathcal{M}_0, \dots, \mathcal{M}_{n-1}\}$. This step can be performed by an external graph partitionner [] and is not adressed by this paper. As a data is mapped onto the mesh, the set of data Δ is partitionned the same way than the mesh in $\{\Delta_0, \dots, \Delta_{n-1}\}$. The second step of the parallelization is to identify in Γ the needed synchronizations between resources to update data, and thus to build a new ordered list of computations Γ_{data}.

Definition For n the number of computations in Γ, and $i < j < n$, a *synchronisation* is needed between c_i and c_j, denoted $c_i \lessdot c_j$, if $c_j = s_j(R_j, w_j, D_j, \exp_j)$, and $w_i \subset R_j$. Moreover, the data to update is $w_i \cap R_j = w_i$.

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. This synchronization is needed because the neighborhood function $\mathcal{N}$ of the stencil computation involves values computed on different resources.

Definition A synchronization between two computations $c_i \lessdot c_j$ is defined as a specific computation $\text{update}(w_i \cap R_j) = \text{update}(w_i)$.

Definition The concatenation of two ordered lists of respectively n and m computations $l_1 = [c_i]_{0 \le i \le n-1}$ and $l_2 = [w_i]_{0 \le i \le m-1}$ is denoted $l_1 \cdot l_2$ and is equal to a new ordered list $l_3 = [c_0, \dots, c_{n-1}, w_0, \dots, w_{m-1}]$.

Definition From the ordered list of computation Γ, a new ordered list Γ_{data} is obtained from the call $\Gamma_{data} = T_{data}(\Gamma, 0)$, where T_{data} is the recursive function defined as

$$T_{data}(\Gamma, i) = \begin{cases} [\Gamma[i]] \cdot T_{data}(\Gamma, i+1) & \text{if } \forall j \le i,\ c_j \not\lessdot c_i \\ [\text{update}(\{w_k\}), \Gamma[i]] \cdot T_{data}(\Gamma, i+1) & \forall k \le i,\ c_k \lessdot c_i \\ [] & \text{if } i = |\Gamma|. \end{cases}$$

The construction of Γ_{data} implies that if a synchronization is needed it is always proceeded before the computation wich needs it. However, an optimization is possible to combine synchronizations together, and thus to synchronize more than one data at a time. If the number of synchronizations is reduced, final performance of the program will be better. For this reason, we define a reduction to apply to Γ_{data}, and which is performed as a post-transformation.

Proposition 2.1 *Denoting c_k and c_j, with $k < j$, two computations of the ordered list Γ_{data} for which a synchronization is needed. If c_i, with $i < j$ is the computation such that $c_i \lessdot c_j$ and such that update(w_i) is performed before c_j in Γ_{data}. And if the needed synchronization before k is update(w_l), where $l < k$. Then, if $i < k$, a single synchronization phase is needed before c_k and is update$(w_l \cup w_i)$.*

Proof By definition, the synchronization of w_i can be performed between c_i and c_j. As $i < k$ the computation c_i is not performed between k and j. As a result, the synchronization of w_i can be performed between c_i and c_k.

In other words, for each synchronization in Γ_{data}, if the computation c_i from which c_j is synchronized occurs before the last synchronization, the two synchronizations can be reduced to one with the union of the data to update.

The final step of this parallelization is to run Γ_{data} on each resource. Thus, for each resource $0 \le k \le p-1$ a multi-stencil program defined by

$$\mathcal{MSP}_k(T, \mathcal{M}_k, \Delta_k, \Gamma_{data}), \tag{1}$$

is runned.

We denote this parallelization technique a coarse-grain data parallelization in contrast with the same technique applied to the finer level of a single computation. In this case, for a computation $c(R, w, D, \exp)$, the local domain D is partitionned for p resources $\{D_0, \dots, D_{p-1}\}$, and each resource k is responsible for a sub-computation $c_k(R, w, D_k, \exp')$. This finer data parallelization technique is not directly adressed by the work presented in this paper, but is exhibited, as it will be explained later.

Task parallelism. A task parallelization technique is a technique to transform a program as a dependency graph of different tasks. A dependency graph exhibits parallel tasks, or on the contrary sequential execution of tasks. In a multi-stencil program a task is defined as a computation $c(R, w, D, \exp)$. As a result, the dependency graph of a multi-stencil program represents the dependencies between local and stencil computations of Γ.

Definition For two computations $c_i(R_i, w_i, D_i, e_i)$ and $c_j(R_j, w_j, D_j, e_j)$, with $i \leq j$, it is said that c_j is dependant from c_i, denoted $c_i \prec c_j$, if $w_i \cap R_j \neq \emptyset$. In this case, c_i has to be computed before c_j. The binary relation $c_i \prec c_j$ represents a *dependency.*

Proposition 2.2 *The binary relation $\prec$ is not transitive.*

Proof Considering three computations $c_0(R_0, w_0, D_0, e_0)$, $c_1(R_1, w_1, D_1, e_1)$ and $c_2(R_2, w_2, D_2, e_2)$, where $w_0 \cap R_1 \neq \emptyset$ and $w_1 \cap R_2 \neq \emptyset$. In this case two dependencies can be extracted $c_0 \prec c_1$ and $c_1 \prec c_2$. However, the dependency $c_0 \prec c_2$ is not true as $w_0 \cap R_2 = \emptyset$.

It has been proved that the binary relation $\prec$ is not transitive. This property is due to the fact that $\prec$ is defined as a data dependency between computations. A dependency chain, however, creates another type of dependency only due to time.

Definition For two computations $c_i(R_i, w_i, D_i, e_i)$ and $c_j(R_j, w_j, D_j, e_j)$, a time dependency is denoted $c_i \blacktriangleleft c_j$ and means that c_i has to be computed before c_j.

Proposition 2.3 *For two computations $c_i(R_i, w_i, D_i, e_i)$ and $c_j(R_j, w_j, D_j, e_j)$ such that $c_i \prec c_j$, $c_i \blacktriangleleft c_j$ is verified.*

Proof By definition $c_i \prec c_j$ means that $w_i \cap R_j \neq \emptyset$ and that c_i has to computed before c_j.

Proposition 2.4 *The binary relation $\blacktriangleleft$ is transitive.*

Proof Considering three computations $c_0(R_0, w_0, D_0, e_0)$, $c_1(R_1, w_1, D_1, e_1)$ and $c_2(R_2, w_2, D_2, e_2)$, where $c_0 \blacktriangleleft c_1$ and $c_1 \blacktriangleleft c_2$. c_0 is computed before c_1 and c_1 is computed before c_2, as a result c_0 is computed before c_2 and the relation $c_0 \blacktriangleleft c_2$ is verified.

Definition A directed acyclic graph (DAG) $G(V, A)$ is a graph where the edges are directed from a source to a destination vertex, and where following the direction of edges, no cycle can be found from a vertex u to itself. A directed edge is called an arc, and for two vertices $v, u \in V$ an arc from u to v is denoted $(\widehat{u, v}) \in A$.

From an ordered list of computations, a directed dependency graph $\Gamma_{task}(V, A)$ can be built finding all pairs of computations $c_i(R_i, w_i, D_i, e_i)$ and $c_j(R_j, w_j, D_j, e_j)$, with $i < j$, such that $c_i \blacktriangleleft c_j$. Those pairs of relations can be found from pairs of data dependencies $c_i \prec c_j$.

Definition From an ordered list of computation $c(R, w, D, \exp)$, a directed dependency graph $\Gamma_{task}(V, A)$ is obtained from the call $T_{task}(\Gamma, 0)$, where T_{task} is the recursive function

$$T_{task}(\Gamma, i, \Gamma_{task}(V, A)) = \begin{cases} (V, A) & \text{if } i = |\Gamma| \\ T_{task}(\Gamma, i+1, \Gamma_{task}(V \cup c_i, A \cup (\widehat{c_k, c_i}))), \forall k \leq i,\ c_k \prec c_i & \text{if } i < |\Gamma| \end{cases}$$

This constructive function is possible because the input is an ordered list. Actually, if $c_k \prec c_i$ then $k < i$. As a result, c_k is already in V when the arc $(\widehat{c_k, c_i})$ is built.

Proposition 2.5 *The directed graph Γ_{task} is an acyclic graph.*

Proof Γ_{task} is built from Γ which is an ordered and sequential list of computations. Moreover, each computation of the list Γ is associated to a vertex of V, even if the same computation is represented more than once in Γ. As a result it is not possible to go back to a previous computation and to create a cycle.

Using the function T_{task} to build Γ_{task}, however, duplication of dependencies may occur because of the transitivity of the relation $\blacktriangleleft$. Actually, as the relation $\prec$ verifies the relation $\blacktriangleleft$, and as $\blacktriangleleft$ is transitive, if for three computations $c_k(R_k, w_k, D_k, e_k)$, $c_i(R_i, w_i, D_i, e_i)$ and $c_j(R_j, w_j, D_j, e_j)$, with $k < i < j$ and where $c_k \prec c_i$, $c_i \prec c_j$ and $c_k \prec c_j$, then the *pseudo-transitivity* $c_0 \blacktriangleleft c_2$ is verified because of the relations $c_0 \blacktriangleleft c_1$ and $c_1 \blacktriangleleft c_2$. However, $c_0 \blacktriangleleft c_2$ is also directly represented by the binary relation $c_0 \prec c_2$. As a result, a duplication of the dependency is created in Γ_{task}. In Figure 3 an example of duplication is given for the relation $c_1 \blacktriangleleft c_4$.

$$\begin{tikzcd}
\boxed{c_0} \arrow[r] & \boxed{c_1} \arrow[r] \arrow[rrr, bend right] & \boxed{c_2} \arrow[r] & \boxed{c_3} \arrow[r] & \boxed{c_4}
\end{tikzcd}$$

Figure 3: Useless duplication of the dependency $c_1 \blacktriangleleft c_4$

Another view of the relations $\prec$ and $\blacktriangleleft$ is that $c_i \blacktriangleleft c_j$ in Γ_{task} is a path from c_i to c_j of any size, while $c_i \prec c_j$ is a path of size 1 from c_i to c_j. And in any case a path in the graph represents a time dependency between computations. Scheduling a graph dependency which do not contains duplication of information is easier. For this reason, we define a reduction to apply to Γ_{task}, and which is performed as a post-transformation. In the example of Figure 3 the dependency represented by the path $c_1 \blacktriangleleft c_2 \blacktriangleleft c_3 \blacktriangleleft c_4$ gives more information than the one directly represented by $c_1 \prec c_4$. The proposed reduction is based on this property.

Definition The dependency DAG $\Gamma_{task}(V, A)$ is reduced to the dependency graph $\Gamma_{task}(V, A')$, where $A' \subseteq A$. A direct arc $(\widehat{c_i, c_j})$, $i < j$, of A, which means that $c_i \prec c_j$, is included in A' if no longer path $c_i \blacktriangleleft c_j$ exists in A because of transitivity.

Hybrid parallelism. It is also possible to combine coarse-grain data and task parallelization techniques to get hybrid parallelism, sometimes more efficient on hybrid architectures. A coarse-grain data parallelism creates for k resources k multi-stencil computations $\mathcal{MSP}_k(T, \mathcal{M}_k, \Delta_k, \Gamma_{data})$, where Γ_{data} is built from an ordered list of computations and is itself an ordered list with the addition of needed synchronizations. On the other hand, the task parallelization technique builds, from an ordered list of computation, a graph dependency Γ_{task}. As a result T_{data} and T_{task} can be composed to build Γ_{hybrid} such as k multi-stencil programs

$$\mathcal{MSP}_k(T, \mathcal{M}_k, \Delta_k, \Gamma_{hybrid})$$

are responsible for an hybrid parallelization of a multi-stencil program. The set of computation Γ_{hybrid} is a DAG of dependencies and synchronizations and can be built from the call to

$$T_{task}(T_{data}(\Gamma, 0), 0, \Gamma_{hybrid}).$$

3 Component model and transformation

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