From DSL to HPC Component-Based Runtime: A Multi-Stencil DSL Case Study

Julien Bigot, Hélène Coullon, Christian Perez

INRIA team Avalon
Maison de la simulation (CEA)

WOLFHPC 2015 - 16th November 2015
Motivation

+ Domain Specific Languages
  - Separation of concerns (domain/implementation)
  - Easy language for the user
  - Implicit optimizations
  - Implicit parallelization

- Domain Specific Languages
  - Difficulties deported to the DSL designer
    - Low level high performance programming
    - Maintainability and portability
  - As many DSLs as domains
Motivation

**Component models**

- Divide an application into several independent black boxes
- Each component defines its interactions with outer world
- Application = Assembly of components

**Component models**

- Maintainability through separation of concerns
- Code-reuse and productivity
- Dynamic assembly of components
Motivation

What if a DSL produces a component-based runtime?

- Is it feasible?
- Is it efficient?
- Does it improve issues of DSLs?
  - maintainability
  - portability
  - productivity

Let’s take a useful example: the *Multi-Stencil Language*!
Multi-Stencil Language

Overview

Compiler

Evaluation

Conclusion and perspectives
Numerical simulation = Multi-Stencil application

Partial Differential Equations
\[
\frac{\partial u(x,y,t)}{\partial t} = \frac{\partial^2 u(x,y,t)}{\partial x^2} + \frac{\partial^2 u(x,y,t)}{\partial y^2}
\]

+ specific behavior for boundary conditions

Time and space discretization

Mesh

Time iterations

Numerical methods

Finite difference/volume/element

Explicit numerical schemes

Stencils

Step by step approximation of the phenomena

Explicit numerical schemes

- Explicit numerical schemes

Julien Bigot (CEA), Hélène Coullon (INRIA), Christian Perez (INRIA)

From DSL to HPC Component-Based Runtime: A Multi-Stencil DSL Case Study
Time and Mesh

Time
At each time iteration of the simulation are applied the *computation kernels* of the application.

Mesh

- A Mesh is a connected undirected graph $\mathcal{M} = (V, E)$ without bridges
- Mesh entities are a subset of $V \cup E$
Data and Computation Kernels

Data
Data is a set of numerical values, each one attached to a given mesh entity

Computation kernel
- Set of data read for the computation
  - Each one associated to a stencil shape
- Data written by the computation
- A numerical expression
- A computation domain
  - Subset of mesh entities
Multi-Stencil Language

Overview

Compiler

Evaluation

Conclusion and perspectives

Multi-Stencil program

\( \text{MSP}(T, M, E, D, \Delta, \Gamma) \)

- \( T \) the set of time iterations to run the simulation
- \( M \) the mesh of the simulation
- \( E \) the set of mesh entities
- \( D \) the set of computation domains
- \( \Delta \) the set of data
- \( \Gamma \) the set of computations

\( = \text{the six sections of a Multi-Stencil Language program!} \)
Example

$\mathcal{MSP}(T, M, E, D, \Delta, \Gamma)$

- **mesh**: cart
- **mesh entities**: cell, edgex
- **computation domains**: allcell in cell, alledgedgex in edgex
- **data**: A, cell, C, edgex
- **time**: 500
- **computations**: $A[\text{allcell}] = \text{comp}(C[n1])$
Multi-Stencil Language

**MSL is not**

- a new stencil optimizer/compiler
- a new distributed data structure

**MSL is**

- a high-level language for multi-stencil simulations
- agnostic from the type of mesh used (data structure)
- based on identifiers only

*MSL produces a ready-to-fill component-based parallel pattern of the simulation!*
Related Work

Complementary work

- Distributed data structures: SkelGIS, Global Arrays
- Stencil DSLs (on grids): Pochoir, PATUS
- Stencil DSLs (on unstructured meshes): OP2, Liszt

Similar work

- Pipeline of stencil computations for image processing: Halide
  - On grids (image), different abstraction level
- DSL to component-based runtime: ?
MSL to Component-based runtime

Ready-to-fill parallel pattern

- Data parallelism
  - External distributed data structure
  - Automatic detection of synchronizations
- Task parallelism (mid-grain)
  - Compile a static scheduling of computation kernels

The fine grain task parallelism is left to other languages:

- OpenMP in the kernels
- Kernels generated by stencil compilers (Pochoir, PATUS, Liszt etc.)
Overview

MSL to Component-based runtime

\[ \mathcal{MSP}(T, M, \mathcal{E}, D, \Delta, \Gamma) \]

- **Driver**
- **DDS** \( M, \mathcal{E} \)
- **Data** \( \Delta, D \)
- **Computations** \( \Gamma \)

Duplicated on each processor/core

Julien Bigot (CEA), Hélène Coullon (INRIA), Christian Perez (INRIA)
MSL to Component-based runtime

\[ MSP(T, M, E, D, \Delta, \Gamma) \]

* Duplicated on each processor/core

Julien Bigot (CEA), Hélène Coullon (INRIA), Christian Perez (INRIA)

From DSL to HPC Component-Based Runtime: A Multi-Stencil DSL Case Study
MSL to Component-based runtime

\[ \mathcal{MSP}(T, \mathcal{M}, \mathcal{E}, \mathcal{D}, \Delta, \Gamma) \]

\[ \text{Driver} \quad \mathcal{DDS} \quad \mathcal{M}, \mathcal{E} \]

\[ \Delta, \mathcal{D} \quad \text{Data} \]

\[ \text{Time} \quad \text{Computations} \quad \Gamma \]

\text{Duplicated on each processor/core}
Example

mesh: cart
mesh entities: cell, edgex, edgey
computation domains:
  allcell in cell
  alledgex in edgex
  alledgey in edgey
  part1edgex in edgex
  part2edgex in edgex
data:
  a, cell
  b, cell
  c, edgex
  d, edgex
  e, edgey

f, cell
g, edgey
h, edgey
i, cell
j, edgey
time: 500
computations:
  b[allcell]=c0(a)
  c[alledgex]=c1(b[n1])
  d[alledgex]=c2(c)
  e[alledgex]=c3(c)
  f[allcell]=c4(d[n1])
  g[alledgex]=c5(e)
  h[alledgex]=c6(f)
  i[allcell]=c7(g, h)
  j[partedgex]=c8(i[n1])
Data parallelism

1. Assembly of components duplicated on each resource
2. External Distributed Data Structure to split data among resources
3. Detect when synchronizations are needed

Synchronization

When a computation read a data, using a stencil shape, that has been written by a previous computation.

\[ \Gamma = [c_0, c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8] \]

\[ \rightarrow [c_0, \text{sync}_1, c_1, c_2, c_3, \text{sync}_4, c_4, c_5, c_6, c_7, \text{sync}_8, c_8] \]
Data and task parallelism

Dependency graph

1. Each node is a computation or a synchronization
2. Each edge is a dependency: a computation reads a data that has been written before.

Dynamic or static scheduling?
Series-Parallel Tree

Valdes & Al, The Recognition of Series Parallel Digraphs, STOC '79

Specific components

- **SEQ** to directly replace $S$ nodes
- **PAR** to directly replace $P$ nodes
- **SYNC** for synchronizations
- **$K$** for computation kernels
Series-Parallel Tree

Valdes & Al, The Recognition of Series Parallel Digraphs, STOC ’79

Specific components

- **SEQ** to directly replace $S$ nodes
- **PAR** to directly replace $P$ nodes
- **SYNC** for synchronizations
- **K** for computation kernels

Loop fusion optimization possible
Component-based runtime

Diagram showing the interactions between Driver, DDS, Data, Time, Computations, SEQ, K(c0), K(c1), K(c7), K(c8), K(c3), K(c5), K(c2), K(c4), K(c6), and SYNC.
Resume

The MSL compiler can produces:

- A data parallel pattern of the multi-stencil application
- An hybrid (data + task) pattern of the multi-stencil application
The MSL compiler can produces:

- A data parallel pattern of the multi-stencil application
- An hybrid (data + task) pattern of the multi-stencil application
Implementation and evaluation

Implementation of MSL: Python, SkelGIS and $L^2 C$

Shallow-water equations: 1 mesh, 3 mesh entities, 7 computation domains, 48 data, 98 computations (32 stencils, 66 local kernels)

Evaluation of the data parallelism

- Full SkelGIS implementation (DDS + specific interfaces to hide communications)
- MSL implementation which uses the SkelGIS DDS
- Thin Nodes TGCC Curie: two 8-cores Intel Sandy Bridge 2.7GHz, 64GB RAM, Infiniband
Evaluations

Mesh size: $10k \times 10k$ Number of iterations: 500

---

Julien Bigot (CEA), Hélène Coullon (INRIA), Christian Perez (INRIA)

From DSL to HPC Component-Based Runtime: A Multi-Stencil DSL Case Study
Conclusion

A DSL for Multi-Stencil applications (MSL)

The compilation of MSL to get data + task parallelism

The dump to a component-based runtime

Data parallelism evaluation: no overhead introduced

Perspectives

Scalability up to 32k cores on TGCC Curie (CEA)

Evaluations of the Data+Task parallelism (OpenMP 3)

Dynamic scheduling (OpenMP 4), CPU+GPU (Pochoir etc.)

Show portability, maintainability introduced by components