Semi-automatic Generation of Adaptive Code

Maxime Schmitt[1], César Sabater[2], Cédric Bastoul[1]

January 23 2017, HiPEAC, IMPACT

[1] University of Strasbourg, Inria
[2] National University of Rosario

Compilation Optimization

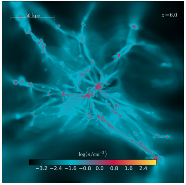
- Compile time
  - e.g. parallelization, vectorization, instruction scheduling ...
  - Extract static informations
  - Generic
- Dynamic optimization (Just-in-time)
  - e.g. thread level speculation, specialized method inlining ...
  - Speculation with possible rollback
  - Specialization at runtime

Sacred rule: Preserve Semantics

Semantics Preservation is not Always Required

What if we don't care about semantics?

Simulation



Multimedia

Relaxing semantics allows new optimizations

- Less precise models
- Less accurate computations
- Adaptive techniques

Our Goal

Provide an API for approximation that is:

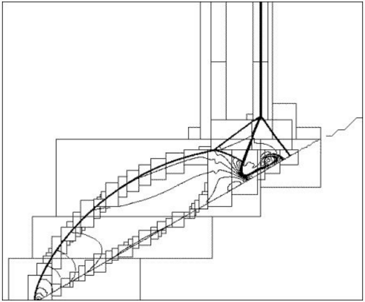
1. Easy to use for the programmer
2. Able to trade accuracy for speed
3. As automatic as possible

Outline

1. Introduction
2. **Adaptive Mesh Refinement**
3. Adaptive Code Refinement
4. Case study: Eulerian fluid simulation
5. Conclusion

Adaptive Mesh Refinement

Change the accuracy of a solution in certain regions, while the solution is being calculated (Berger et al. 1989)



AMR grid for a shock impacting an inclined slope

- More nested $\iff$ more refined
- Ignore regions without shock
- The grid is dynamic and evolves with the simulation

Adaptive Code Refinement

Provide adaptivity without requiring the programmer to heavily modify his code

- Exploits domain-specific knowledge
- Adapts computation dynamically
- Automatically generated adaptive code

And at the compiler level:

- Get information from user pragmas
- Use a runtime to adapt the computations
- Generate specialized code with polyhedral toolkit

ACR User Pragmas

#pragma acr grid(size)

Granularity of the adaptive grid

#pragma acr monitor(data, collector)

Input data for decision mechanism

#pragma acr alternative name(type, effect)

Specifies a way of providing adaptivity to cells

#pragma acr strategy(criteria, alternative)

Specifies in which case an alternative has to be used

Code without ACR

Need of adaptivity for this kernel called at each simulation step

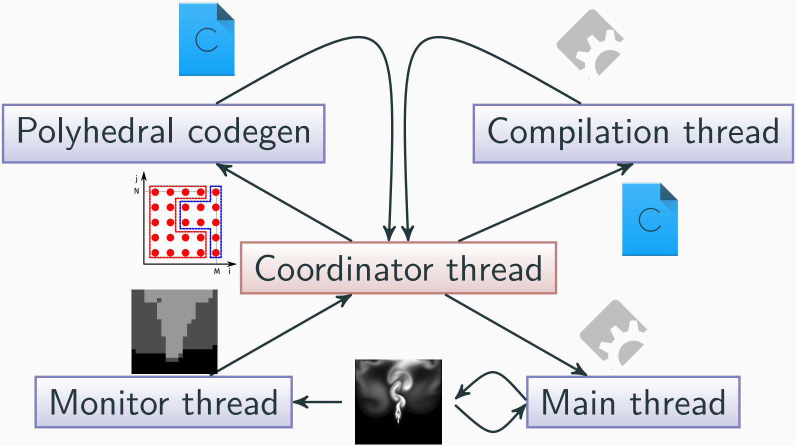
```
for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      computation(k, i, j, density, ...);
```

Code with ACR

The pragmas relax the semantics for the following code block

```
#pragma acr grid(2)
#pragma acr monitor(density[j][i], cell_collector)
#pragma acr alternative high(parameter, solver_steps = 10)
#pragma acr alternative low(parameter, solver_steps = 1)
#pragma acr strategy direct(0, high)
#pragma acr strategy direct(1, low)
for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      computation(k, i, j, density, ...);
```

ACR Runtime



Dedicated threads to gather informations and generate adaptive code

Extracting Runtime Informations

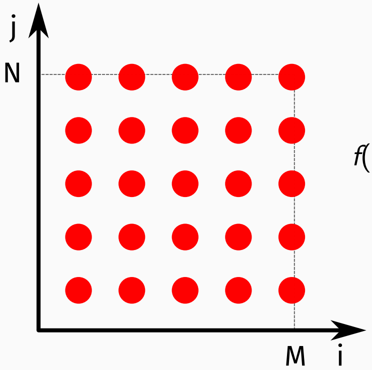
What is the relevant data?

```
#pragma acr grid(2)
#pragma acr monitor(density[j][i], cell_collector)
#pragma acr alternative high(parameter, solver_steps = 10)
#pragma acr alternative low(parameter, solver_steps = 1)
#pragma acr strategy direct(0, high)
#pragma acr strategy direct(1, low)
for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      computation(k, i, j, density, ...);
```

#pragma acr monitor(data, collector)

Input data for decision mechanism

Iteration Domain and Access Function

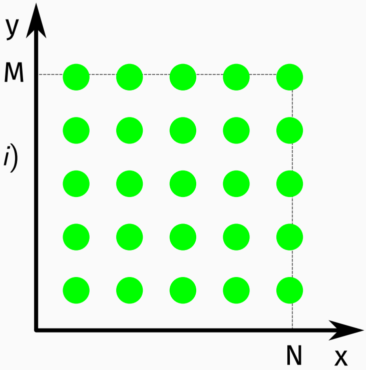


Iteration domain

access

$f(i, j) = (j, i)$

$\longrightarrow$



Data space

Extracting Static Grid Size

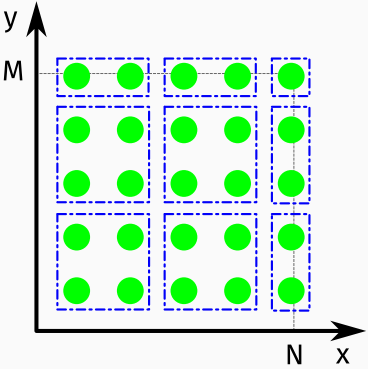
How much granularity is required?

```
#pragma acr grid(2)
#pragma acr monitor(density[j][i], cell_collector)
#pragma acr alternative high(parameter, solver_steps = 10)
#pragma acr alternative low(parameter, solver_steps = 1)
#pragma acr strategy direct(0, high)
#pragma acr strategy direct(1, low)
for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      computation(k, i, j, density, ...);
```

#pragma acr grid(size)

Granularity of the adaptive grid

Iteration Domain and Access Function



The data domain is tiled with respect to the grid pragma

Extracting the Criteria

How to collect information for each tile?

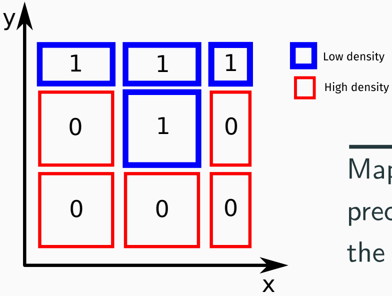
```
#pragma acr grid(2)
#pragma acr monitor(density[j][i], cell_collector)
#pragma acr alternative high(parameter, solver_steps = 10)
#pragma acr alternative low(parameter, solver_steps = 1)
#pragma acr strategy direct(0, high)
#pragma acr strategy direct(1, low)
for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      computation(k, i, j, density, ...);
```

#pragma acr strategy(criteria, alternative)

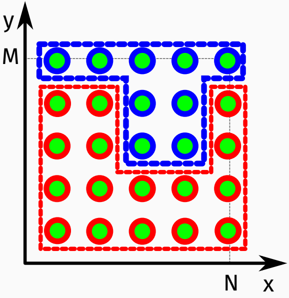
Specifies in which case an alternative has to be used

Precision Level Mapped on the Data Space

Each tile gets a precision level assigned by a monitoring thread



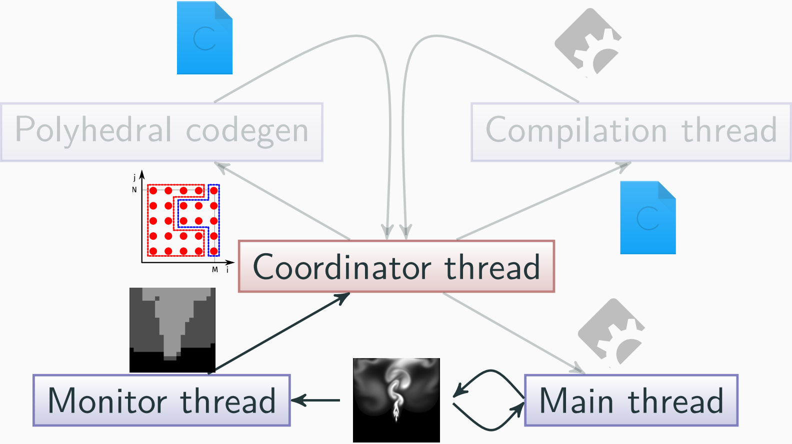
Mapping the precision over the data space



Precision matrix

Precision for each data coordinates

Runtime Information Extraction



Extracting Adaptivity Information

Now that the precision matrix is known, what to do for each precision level?

```
#pragma acr grid(2)
#pragma acr monitor(density[j][i], cell_collector)
#pragma acr alternative high(parameter, solver_steps = 10)
#pragma acr alternative low(parameter, solver_steps = 1)
#pragma acr strategy direct(0, high)
#pragma acr strategy direct(1, low)
  for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      computation(k, i, j, density, ...);
```

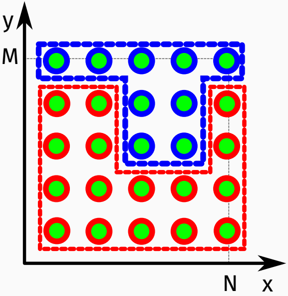
#pragma acr alternative name(type, effect)

Specifies a way of providing adaptivity to cells

Code with Runtime Test

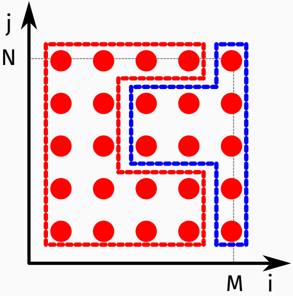
Generic implementation of the kernel

```
for (int k = 0; k < solver_steps; ++k)
  for (int i = 0; i <= M; ++i)
    for (int j = 0; j <= N; ++j)
      if (computation_needed(k, i, j))
        computation(k, i, j, density, ...);
```



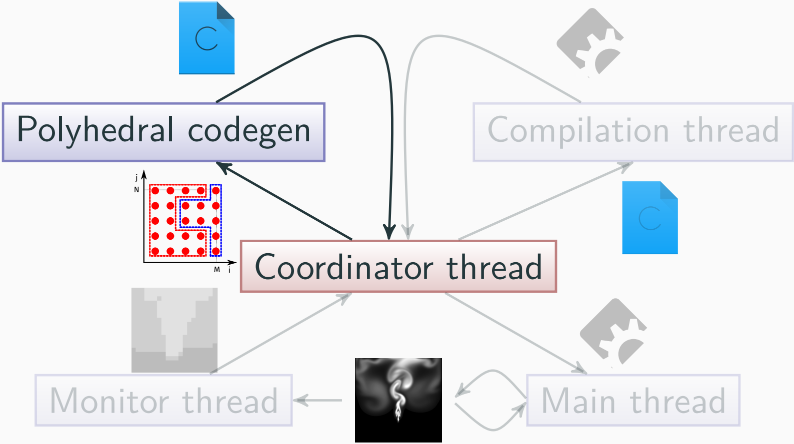
Precision polyhedra
(Red=high , Blue=low)

Preimage of the computed polyhedra $\longrightarrow$



Apply the restriction to each polyhedron

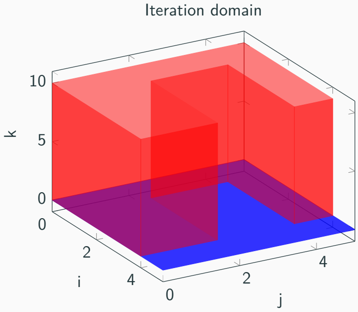
From Polyhedral Representation Back to Code



Code With CLooG

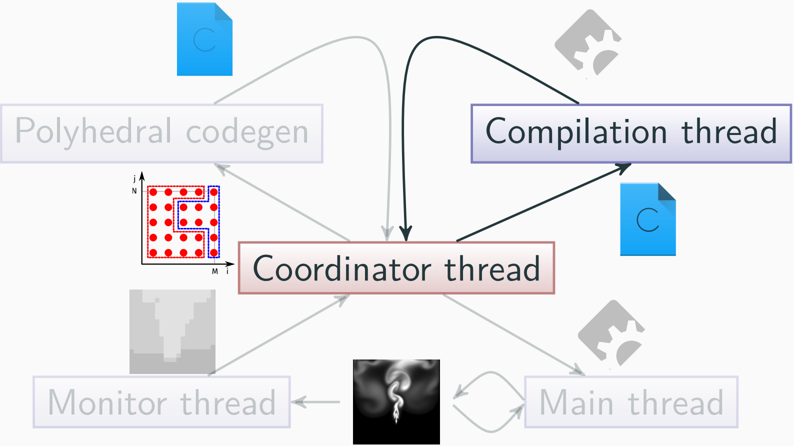
Same code generated with CLooG

```
for (i=0;i<=5;i++)
   for (j=0;j<=5;j++)
      computation(0,i,j,...);
for (k=1;k<=10;k++) {
   for (i=0;i<=2;i++)
      for (j=0;j<=5;j++)
         computation(k,i,j,...);
   for (i=3;i<=4;i++) {
      for (j=0;j<=1;j++)
         computation(k,i,j,...);
      computation(k,i,5,...);
   }
}
```



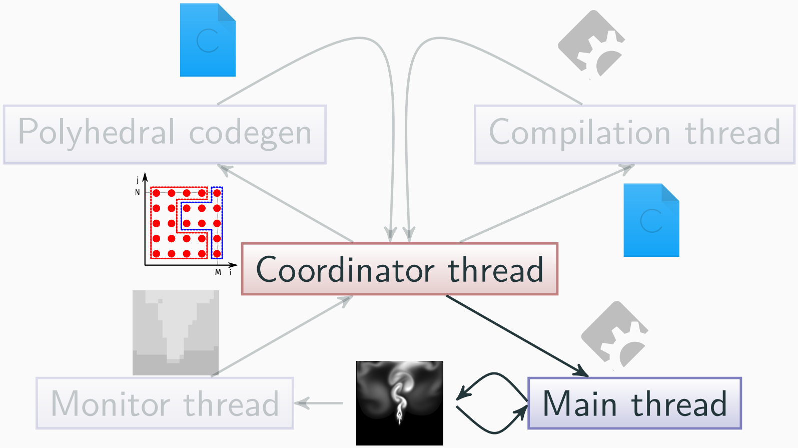
(Red = high , Blue = low)

Generate the Adaptive Code



Compilation using TCC and GCC -O2

Code injection



Outline

Case study: Eulerian fluid simulation

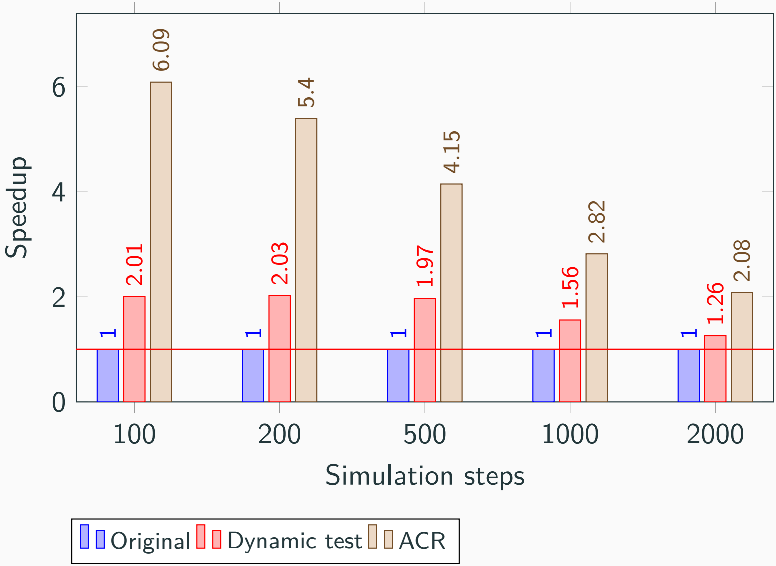
Graphical representation of the simulation fluid density for a test simulation

- Simulation of fluid behaviour over time
- Grid based simulation
  - Fluid density
  - Fluid velocity
- The solver converges with an iterative algorithm
  - Higher density requires more solver iterations

Live demo !

What could possibly go wrong ?

Simulation speedup



Precision error

The fluid density value ranges from 0 to 20

Sim steps	Mean error	Max error	Raw value	Raw % total
100	0.13%	0.41%	0.002	0.01%
200	0.15%	11.14%	0.004	0.02%
500	0.36%	25.41%	0.02	0.1%
1000	1.60%	30.58%	0.04	0.2%
2000	3.70%	55.4%	0.08	0.4%

Conclusion

- New compiler technique to generate adaptive code
  - Automatic with user input informations
  - Easy to use and keeps the code simple
  - Asymetric threading model
- Future work
  - Automatic grid tuning
  - Provide a way to ensure a certain precision
  - Explore other approximation techniques

Questions ?

Image reference

- Simulation: https://commons.wikimedia.org/wiki/File:Views_of_a_simulated_primordial_galaxy,_density_map.png
- Cat: https://commons.wikimedia.org/wiki/File:Felis_silvestris_silvestris_small_gradual_decrease_of_quality.png
- AMR: https://commons.wikimedia.org/wiki/File:Amrgridimg.jpg