Semi-automatic Generation of Adaptive Code

Maxime Schmitt\textsuperscript{1}, César Sabater\textsuperscript{2}, Cédric Bastoul\textsuperscript{1}

January 23 2017, HiPEAC, IMPACT

\textsuperscript{1} University of Strasbourg, Inria
\textsuperscript{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
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)

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

AMR grid for a shock impacting an inclined slope
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
#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
Need of adaptivity for this kernel called at each simulation step

```c
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, ...);
```
The pragmas relax the semantics for the following code block

```c
#pragma a c r    grid (2)
#pragma a c r    monitor (density [ j ][ i ], cell _ collector )
#pragma a c r    alternative high (parameter, solver _ steps = 10)
#pragma a c r    alternative low (parameter, solver _ steps = 1)
#pragma a c r    strategy direct (0, high )
#pragma a c r    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

Polyhedral codegen

Compilation thread

Coordinator thread

Monitor thread

Main thread

Dedicated threads to gather informations and generate adaptive code
What is the relevant data?

```c
#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, ...);
```
Iteration Domain and Access Function

Iteration domain

Data space

access

\[ f(i, j) = (j, i) \]
How much granularity is required?

```c
#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
The data domain is tiled with respect to the grid pragma.
Extracting the Criteria

How to collect information for each tile?

1. #pragma acr grid(2)
2. #pragma acr monitor(density[j][i], cell_collector)
3. #pragma acr alternative high(parameter, solver_steps = 10)
4. #pragma acr alternative low(parameter, solver_steps = 1)
5. #pragma acr strategy direct(0, high)
6. #pragma acr strategy direct(1, low)
7. 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
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

- Coordinator thread
- Compilation thread
- Polyhedral codegen
- Monitor thread
- Main thread
Now that the precision matrix is known, what to do for each precision level?

1. `#pragma acr grid(2)`
2. `#pragma acr monitor(density[j][i], cell_collector)`
3. `#pragma acr alternative high(parameter, solver_steps = 10)`
4. `#pragma acr alternative low(parameter, solver_steps = 1)`
5. `#pragma acr strategy direct(0, high)`
6. `#pragma acr strategy direct(1, low)`
7. ```
   for (int k = 0; k < solver_steps; ++k)
   ```
8. ```
   for (int i = 0; i <= M; ++i)
   ```
9. ```
   for (int j = 0; j <= N; ++j)
   ```
10. `computation(k, i, j, density, ...);`

`#pragma acr alternative name(type, effect)`

Specifies a way of providing adaptivity to cells
Generic implementation of the kernel

```c
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, ...);
```
Reconstruction of the Iteration Domain

Preimage of the computed polyhedra

Precision polyhedra
(\text{Red} = \text{high}, \text{Blue} = \text{low})

Apply the restriction to each polyhedron
From Polyhedral Representation Back to Code

Coordinator thread

Compilation thread

Polyhedral codegen

Main thread

Monitor thread
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, ...);
    }
}
Generate the Adaptive Code

Polyhedral codegen

Compilation thread

Coordinator thread

Monitor thread

Main thread

Compilation using TCC and GCC -O2
Code injection

Polyhedral codegen

Compilation thread

Coordinator thread

Monitor thread

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

Graphical representation of the simulation fluid density for a test simulation
Live demo!

What could possibly go wrong?
Simulation speedup

Simulation steps

Speedup

Original  Dynamic test  ACR

100  1  2.01  6.09
200  1  2.03  5.4
500  1  1.97  4.15
1000  1  1.56  2.82
2000  1  1.26  2.08
The fluid density value ranges from 0 to 20

<table>
<thead>
<tr>
<th>Sim steps</th>
<th>Mean error</th>
<th>Max error</th>
<th>Raw value</th>
<th>Raw % total</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.13%</td>
<td>0.41%</td>
<td>0.002</td>
<td>0.01%</td>
</tr>
<tr>
<td>200</td>
<td>0.15%</td>
<td>11.14%</td>
<td>0.004</td>
<td>0.02%</td>
</tr>
<tr>
<td>500</td>
<td>0.36%</td>
<td>25.41%</td>
<td>0.02</td>
<td>0.1%</td>
</tr>
<tr>
<td>1000</td>
<td>1.60%</td>
<td>30.58%</td>
<td>0.04</td>
<td>0.2%</td>
</tr>
<tr>
<td>2000</td>
<td>3.70%</td>
<td>55.4%</td>
<td>0.08</td>
<td>0.4%</td>
</tr>
</tbody>
</table>


**Conclusion**

- New compiler technique to generate adaptive code
  - Automatic with user input informations
  - Easy to use and keeps the code simple
  - Asymmetric threading model

- Future work
  - Automatic grid tuning
  - Provide a way to ensure a certain precision
  - Explore other approximation techniques
Questions ?
- 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