Data-reuse Optimizations for Pipelined Tiling with Parametric Tile Sizes

Alexandre Isoard
ENS de Lyon, France
Ph.D advisor: Alain Darte

Polyhedral Model

Symbolic representation of the semantic of affine kernels.
Allows arbitrary nesting/sequence of `for` loops and `if` conditionals, data manipulation exclusively through (possibly multidimensional) arrays. Requires all predicates (loop bounds and conditional statements) as well as all array accesses to be affine expressions of parameters or surrounding loop counters. Thus insuring the full static analyzability of the kernel using integer linear programming.
☛ Linear algebra, stencil code, some graph algorithm, statistics, ...

Matrix Multiply

```
for(int i = 0; i < n; ++i) {
   for(int j = 0; j < n; ++j) {
S:    C[i][j] = 0;
      for(int k = 0; k < n; ++k)
T:       C[i][j] += A[i][k] * B[k][j];
   }
}
```

Polyhedral Description

```
Domain := [n] -> { S[i][j]    : 0 <= i,j < n;
                   T[i][j][k] : 0 <= i,j,k < n }
Read := [n] -> { T[i][j][k] -> A[i][k];
                 T[i][j][k] -> B[k][j];
                 T[i][j][k] -> C[i][j]; }
Write := [n] -> { S[i][j]    -> C[i][j];
                  T[i][j][k] -> C[i][j]; }
Schedule := [n] -> { S[i][j]    -> [i][j][0][0];
                     T[i][j][k] -> [i][j][1][k]; }
```

`Schedule` maps the iteration space to a logical time space ordered by the lexicographic order.

Parametric Tiling

Tiling is a common technique involving reordering of computation into blocks.
The objective is to increase the locality of the data accesses. It usually require a rescheduling of the loop-nest prior to blocking to insure correctness but also to improve the final locality. Selection and application of a good schedule is made possible and fully automatic by current polyhedral techniques.

Polynomial Product

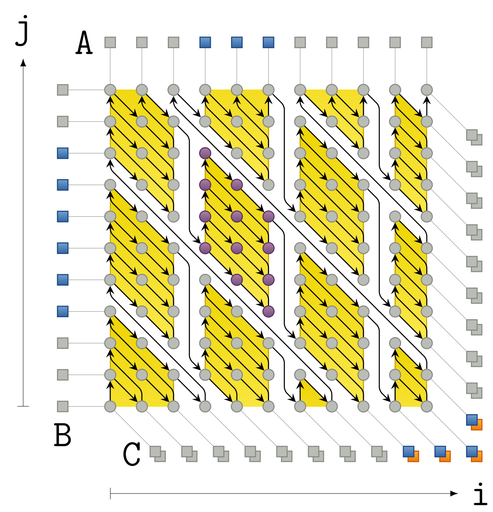
```
for(int k = 0; k < 2*n-1; ++k)
S: C[k] = 0;
for(int i = 0; i < n; ++i)
   for(int j = 0; j < n; ++j)
T:    C[i+j] += A[i] * B[j];
```

Optimal Schedule

```
for(int k = 0; k < 2*n-1; ++k) {
S: C[k] = 0;
   for(int i = max(0,k+1-n); i <= min(k,n-1); ++i)
T:    C[k] += A[i] * B[k-i];
}
```

When the size of the generated tiles is a parameter instead of a static constant we speak of parametric tiling.
Parametric tiling is problematic as it breaks the polyhedral model by introducing a division or a modulo between unknowns. Problem which is usually dealt by applying the transformation at the very end of the optimization process, during code generation. Thus forbidding any further optimization involving inter-tiles analysis.

Tiled Polynomial Product



Execution of the given tile will require to:
- load 3 cells of `A`,
- load 6 cells of `B`,
- load/store 4 cells of `C`,

to compute 12 iterations with a parallelism of 4.

Local memory will host at most:
- 3 cells of `A`,
- 6 cells of `B`,
- 4 cells of `C`.

Kernel Offloading / Scratch Pad Programming / Instruction Level Parallelization

Kernel Offloading

Exporting a computationally expensive piece of code to an external accelerator.
☛ e.g.: FPGA, GPU, ASICs

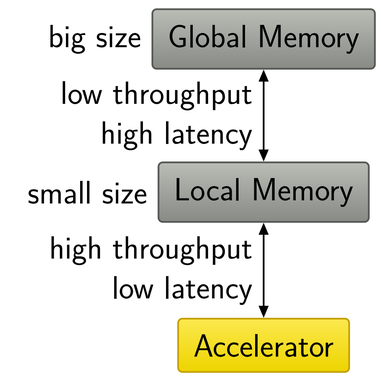
Scratch Pad Programming

Involves an explicit cache. Transfers from and to the cache are governed by the software through explicit copies.
☛ e.g.: GPU Shared Memory

Instruction Level Parallelization

When enough spare registers are available, increasing the number of parallel instructions improves performances.
☛ e.g.: GPU Private Memory

All those techniques involve more or less the same scheme:



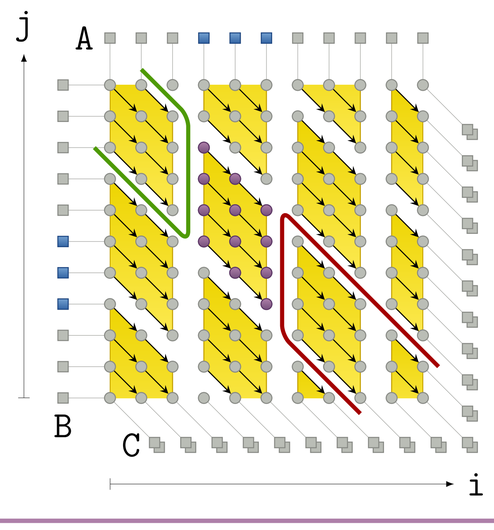
Dealing with all of them at the same time can be achieved through recursive tiling. But the parameter space of tile sizes becomes so big that exploration is out of reach, hence the need for a parametric analysis.

Data-Reuse / Pipelining

Executing the kernel tile by tile and holding and transferring only necessary data allows to control the local memory footprint and the amount of communication. Memory transfers are coalesced before and after each tile and benefit from the good locality. Pipelining of the computation consists in loading the data for the current tile during the computation of the previous ones and in storing back the results during the computation of the next ones. **Pipelining requires a full data-reuse on at least the same length** to avoid incorrectly loading data while it is being produced by running tiles.

As a rule of thumb:
coalescing on bigger tiles → less memory transfer, bigger local memory
pipelining with a bigger pipeline depth → better transfer/computation overlap
data-reuse with a bigger reuse length → less memory transfer, bigger local memory

Parametric Reuse Analysis



Main trick for Parametric Analysis:
The set of points to the left of the green line and the set of points to the right of the red line have a computable affine expression! Knowing the tile in which those points actually live is not required[1].

Execution of the given tile will require to:
- load 3 cells of `A`,
- load 3 cells of `B`,

to compute 12 iterations with a parallelism of 4.

In case of double buffering (pipeline of depth 2), local memory size will be at most:
- 6 cells of `A`,
- 9 cells of `B`,
- 4 cells of `C`.

What is done? / What is next?

- Theory for the exact and approximated case is published:
  A. Darte and A. Isoard. "Parametric tiling with inter-tile data reuse." in IMPACT'14, 2014
- Proof of concept implementation of the exact case analysis is working.
- A basic lattice based memory allocation has been implemented.
- Full implementation into ppcg is underway.
- Cost model to actually choose the tile size remains to do.
- Designing a general scheme for approximation of data sets might be interesting.
- Improved lattice based memory allocation.

http://www.cnrs.fr/ http://www.inria.fr/ http://www.ens-lyon.eu/ http://www.ens-lyon.fr/LIP/ http://www.univ-lyon1.fr/ http://www.universite-lyon.fr/