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, ...

Kernel Offloading / Scratch Pad Programming / Instruction Level Parallelization

All those techniques involve more or less the same scheme:  
- big size Global Memory  
- low throughput high latency  
- small size Local Memory  
- high throughput low latency Accelerator

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.

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 requires 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.

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.

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 communications. 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

What is done? / What is next?

- Theory for the exact and approximated case is published:  
- 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.