Data-reuse Optimizations for Pipelined Tiling with Parametric Tile Sizes
Alexandre Isoard

To cite this version:
Alexandre Isoard. Data-reuse Optimizations for Pipelined Tiling with Parametric Tile Sizes. 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT’14), Aug 2014, Edmonton, Canada. hal-01111393

HAL Id: hal-01111393
https://inria.hal.science/hal-01111393
Submitted on 30 Jan 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

Copyright
### 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. Includes Linear algebra, stencil code, some graph algorithm, statistics, ...

### Data-reuse Optimizations for Pipelined Tiling with Parametric Tile Sizes

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

### Polyhedral Description

Matrix Multiply

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

### Parameterized 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 an effective schedule is made possible and fully automatic by current polyhedral techniques.

### Polyhedral Product

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

### Polyhedral Schedule

```c
for (int i = max (0 ,k+1 -n); i <= min (k,n -1); ++i) {
    ...}
```

To compute 12 iterations with a parallelism of 4. Execution of the given tile will require to:
- load 3 cells of A, L
- load 8 cells of B, L
- 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, L
  - 6 cells of B, L
  - 4 cells of C, L

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

### 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 overhead. 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 one and in storing back the results during the computation of the next one. 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.

Execution of the given tile will require to:
- load 3 cells of A, L
- load 8 cells of B, L
- load/store 4 cells of C, 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, L
  - 9 cells of B, L
  - 4 cells of C, L

### Kernel Offloading

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

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

### 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

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.

### Instruction Level Parallelization

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