Automatic High-Performance GPU code Generation using CUDA-CHiLL

Malik Khan (mmurtaza@usc.edu)
Jacqueline Chame (jchame@isi.edu)
Gabe Rudy, Chun Chen, Mary Hall, Mark Hall
{grudy,chunchen,mhall,markhall}@cs.utah.edu

Funded by NSF award CCR-0615412, and DOE grant DE-FC02-06ER25765.

CUDA-CHiLL Transformations

<table>
<thead>
<tr>
<th>Code Example</th>
<th>2D Convolution: CUDA-CHiLL recipe and optimized code</th>
</tr>
</thead>
</table>

Matrix-Vector Multiply

- **CUDASChiLL Recipe**
  - Data with reuse inside threads, mapped to registers
  - Data with reuse across threads, mapped to shared memory
  - Data with non-coalesced global memory accesses, mapped to shared memory.

**Manage Heterogeneous Memory Hierarchy**
- Data with reuse inside threads, mapped to registers
- Data with reuse across threads, mapped to shared memory
- Data with non-coalesced global memory accesses, mapped to shared memory.

**Other Optimizations**
- Aggressive loop unrolling to:
  - Improve ILP
  - Increase register reuse
  - Reduce loop overhead.

**Optimized Code**
- _shared_float_F (i); _shared_float_F (j); float tmp;
- for (mp = 16 * by + 3 * ty; mp < min(16 * by + 30, 16 * by + 3 * ty + 2); mp++)
  - for (tx1 = 2 * tx; tx1 <= min(2 * tx + 1, 46); tx1+)
  - for (ty = 0; ty <= 15; ty++)
    - _P1[mp + ty * (ti + 2)] = b[mp + ty * (ti + 2)];
    - _P2[mp + ty] = a[mp + ty * ti];
  - for (k = 0; k <= 15; k++)
    - _P1[mp + ty * (ti + 2)] = b[mp + ty * (ti + 2)];
    - _P2[mp + ty] = a[mp + ty * ti];
  - for (l = 0; l <= 15; l++)
    - _P1[mp + ty * (ti + 2)] = b[mp + ty * (ti + 2)];
    - _P2[mp + ty] = a[mp + ty * ti];
  - for (tx1 = 2 * tx; tx1 <= min(2 * tx + 1, 46); tx1+)
    - for (ty = 0; ty <= 15; ty++)
      - _P1[mp + ty * (ti + 2)] = b[mp + ty * (ti + 2)];
      - _P2[mp + ty] = a[mp + ty * ti];

Performance Comparison

- **Matrix-Matrix Multiplication**
- **2D Convolution**
- **Matrix Transpose**

Motivation

- A Compiler-based Transformation, Code Generation and Auto-tuning System, which
  - Applies common compiler transformations.
  - Finds optimal computation mapping heuristics.
  - Searches high-optimized code for a target GPU
- Goal: Achieve performance comparable to manually tuned code

CUDA-CHiLL

- Input: Sequential Loop nest computation
- Optimizing decisions
- Computational decomposition
- Data Staging
- Polychromatic
- Transformation and code generation
- Cudaize transform
- Autotuning
- Output: CUDA code

Decision Algorithm

- Purpose: Automatically derive transformation recipes for sequential loop nest computations (affine)
- Parallel Mapping: Select candidates for block and threads, through:
  - Dependence analysis & Global memory coalescing
  - Generate tiling for block and thread decomposition.
- Performance Comparison
- **Matrix-Vector Multiplication**
- **2D Convolution**
- **Matrix Transpose**

Code Example

- **CUDASChiLL Transformations**
- **CUDA Code**

**CUDA-CHiLL Recipe**

- _global_.GPU_MV(float a, float b, float** c) {
  int i = 32 * bx + tx;
  int bx = blockIdx.x; int tx = threadIdx.x;
  __global__ GPU_MV(float* a, float* b, float** c) {
    for (j = 0; j < N; j++) {
      float acpy = a[tx + 32 * bx];
      __shared__ float bcpy[32];
      for (tmp = 0; tmp <= 15; tmp++)
        bcpy[tx] = b[32 * jj + tx];
    }
    __syncthreads();
    for (j = 32 * jj; j <= 32 * jj + 32; j++)
      __syncthreads();
    c[k + 16 * by][tx + 32 * bx] = tmp3;
    tmp3 = c[k + 16 * by][tx + 32 * bx];
  }
  for (tmp = 16 * by + 3 * ty; tmp <= min(16 * by + 30, 16 * by + 3 * ty + 2); tmp++)
    for (tx1 = 2 * tx; tx1 <= min(2 * tx + 1, 46); tx1+)
      for (by = 0; by <= 15; by++)
        _P1[tmp][tx] = b[tmp][tx];
    for (ty = 0; ty <= 15; ty++)
      _P2[tmp] = a[tmp][by];
    __syncthreads();
    for (k = 0; k <= 15; k++)
      _P1[tmp][tx] = b[tmp][tx];
      _P2[tmp] = a[tmp][by];
    __syncthreads();
}

- **Copy Code to Registers**
- **Copy Code to Shared Memory**
- **Copy Code to Global Memory**
- **Copy Code to Registers + Local Memory**