Optimizing Stencil Computations

March 18, 2013

Administrative

Midterm coming

- April 3?
- In class March 25, can bring one page of notes
- Review notes, readings and review lecture
- Prior exams are posted

Design Review

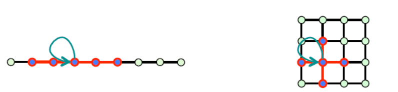
- Intermediate assessment of progress on project, oral and short
- In class on April 1

Final projects

- Poster session, April 24 (dry run April 22)
- Final report, May 1

Stencil Computations

A stencil defines the value of a grid point in a d-dimensional spatial grid at time t as a function of neighboring grid points at recent times before t.



(a) 1D 5-Point Stencil (b) 2D 5-Point Stencil

Stencil Computations, Performance Issues

- Bytes per flop ratio is O(1)
- Most machines cannot supply data at this rate, leading to memory bound computation
- Some reuse, but difficult to exploit fully, and interacts with parallelization

How to maximize performance:

- Avoid extraneous memory traffic, such as cache capacity/conflict misses
- Bandwidth optimizations to maximize utility of memory transfers
- Maximize in-core performance

Learn More: StencilProbe

- See http://people.csail.mit.edu/skamil/projects/stencilprobe/
- Several variations of Heat Equation, to be discussed
- Can instantiate to measure performance impact

Example: Heat Equation

```
for (t=0; t<timesteps; t++)  {  // time step loop
  for (k=1; k<nz-1; k++) {
    for (j=1; j<ny-1; j++) {
      for (i=1; i<nx-1; i++) {
        // 3-d 7-point stencil
        B[i][j][k] = A[i][j][k+1] + A[i][j][k-1] +
             A[i][j+1][k] + A[i][j-1][k] + A[i+1][j][k] +
             A[i-1][j][k] – 6.0 * A[i][j][k] / (fac*fac);
      }
    }
  }
  temp_ptr = A;
  A = B;
  B = temp_ptr;
}
```

What if nx, ny, nz large?

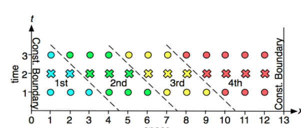
Heat Equation, Add Tiling

```
for (t=0; t<timesteps; t++)  {  // time step loop
  for (jj = 1; jj < ny-1; jj+=TJ) {
    for (ii = 1; ii < nx - 1; ii+=TI) {
      for (k=1; k<nz-1; k++) {
        for (j = jj; j < MIN(jj+TJ,ny - 1); j++) {
          for (i = ii; i < MIN(ii+TI,nx - 1); i++) {
            // 3-d 7-point stencil
            B[i][j][k] = A[i][j][k+1] + A[i][j][k-1] +
                A[i][j+1][k] + A[i][j-1][k] + A[i+1][j][k] +
                A[i-1][j][k] – 6.0 * A[i][j][k] / (fac*fac);
          }
        }
      }
  temp_ptr = A;
  A = B;
  B = temp_ptr;
}
```

Note the reuse across time steps!

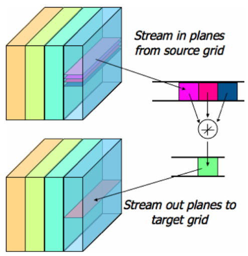
Heat Equation, Time Skewing

```
for (kk=1; kk < nz-1; kk+=tz) {
  for (jj = 1; jj < ny-1; jj+=ty) {
    for (ii = 1; ii < nx - 1; ii+=tx) {
      for (t=0; t<timesteps; t++)  {  // time step loop
        … calculate bounds from t and slope …
        for (k=blockMin_z; k < blockMax_z; k++) {
          for (j=blockMin_y; j < blockMax_y; j++) {
            for (i=blockMin_x; i < blockMax_x; i++) {
              // 3-d 7-point stencil
              B[i][j][k] = A[i][j][k+1] + A[i][j][k-1] +
                  A[i][j+1][k] + A[i][j-1][k] + A[i+1][j][k] +
                  A[i-1][j][k] – 6.0 * A[i][j][k] / (fac*fac);
            }
          }
        }
  temp_ptr = A;
  A = B;
  B = temp_ptr;
}
```



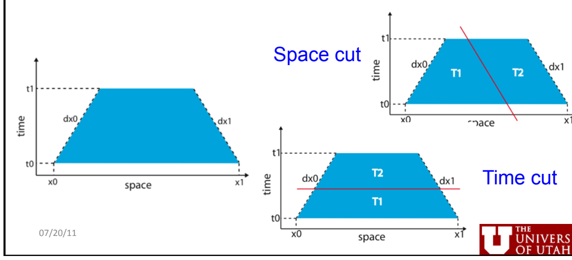
Heat Equation, Circular Queue

- See probe_heat_circqueue.c



Heat Equation, Cache Oblivious

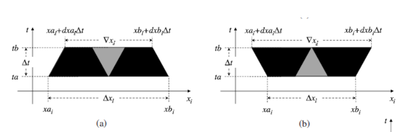
- See probe_heat_oblivious.c
- Idea: Recursive decomposition to cutoff point
- Implicit tiling: in both space and time
- Simpler code than complex tiling, but introduces overhead
- Encapsulated in Pochoir DSL (next slide)



Example Pochoir Stencil Compiler Specification

```
#define mod(r,m) ((r)%(m) + ((r)<0)? (m):0)

Pochoir_Boundary_2D(heat_bv, a, t, x, y)
  return a.get(t,mod(x,a.size(1)),mod(y,a.size(0)));
Pochoir_Boundary_End

int main(void) {

  Pochoir_Shape_2D 2D_five_pt[] = {{1,0,0}, {0,1,0},
        {0,-1,0}, {0,-1,-1}, {0,0,-1}, {0,0,1}};
  Pochoir_2D heat(2D_five_pt);

  Pochoir_Array_2D(double) u(X, Y);
  u.Register_Boundary(heat_bv);
  heat.Register_Array(u);

  Pochoir_Kernel_2D(heat_fn, t, x, y)
    u(t+1, x, y) = CX * (u(t, x+1, y) - 2 * u(t, x,
          y) + u(t, x-1, y)) + CY * (u(t, x, y+1) - 2
          * u(t, x, y) + u(t, x, y-1)) + u(t, x, y);
  Pochoir_Kernel_End

  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      u(0, x, y) = rand();

  heat.Run(T, heat_fn);

  for (int x = 0; x < X; ++x)
    for (int y = 0; y < Y; ++y)
      cout << u(T, x, y);


  return 0;
}
```


Parallel Stencils in Pochoir



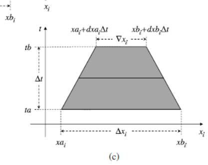
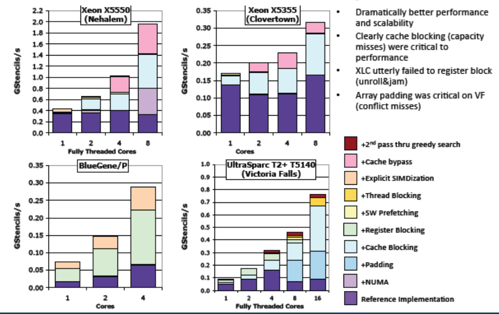


Figure 7: Cutting projection trapezoids. The spatial dimension increases to the right, and the time runs upward. (a) Trisecting an upright trapezoid using a parallel space cut produces two black trapezoids that can be processed in parallel and a gray trapezoid that must be processed after the black ones. (b) Trisecting an inverted trapezoid using a parallel space cut produces two black trapezoids that can be processed in parallel and a gray trapezoid that must be processed before the black ones. (c) A time cut produces a lower and an upper trapezoid where the lower trapezoid must be processed before the upper.

General Approach to Parallel Stencils

- Always safe to parallelize within a time step
- Circular queue and time skewing encapsulate "tiles" that are independent

Results for Heat Equation



Reference: K. Datta et al., "*Stencil Computation Optimization and Autotuning on State-of-the-Art Multi-core Architectures",* SC08.

07/20/11

What about GPUs?

- Two recent papers:

"Auto-Generation and Auto-Tuning of 3D Stencil Codes on GPU Clusters," Zhang and Mueller, CGO 2012.

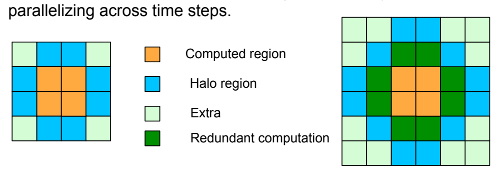
"High-Performance Code Generation for Stencil Computations on GPU Architectures," Holewinski et al., ICS 2012.

- Key issues:

—Exploit reuse in shared memory.
—Avoid fetching from global memory.
—Thread decomposition to support global memory coalescing.

Overlapped Tiling for Halo Regions (or Ghost Zones)

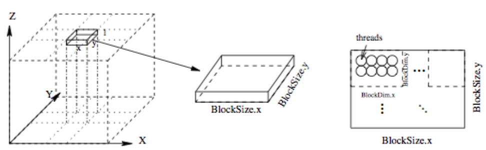
- Input data exceeds output result (as in Sobel)
- Halo region or ghost zone extends the per-thread data decomposition to encompass additional input
- An (n+2)x(n+2) halo region is needed to compute an nxn block if subscript expressions are of the form ± 1, for example
- By expanding the halo region, we can trade off redundant computation for reduced accesses to global memory when parallelizing across time steps.



2-d 5-point stencil example

2.5D Decomposition

• Partition such that each thread block sweeps over the z-axis and processes one plane at a time.



(a) Decompose Space Into Columns

(b) Column Size is (BlockSize.x, BlockSize.y, Z)

(c) Unroll on Both X and Y Dimensions

Resulting Code

```
#define sizey (BLOCK_Y+halo_j*2)
#define sizex (BLOCK_X+halo_i*2)

template <class T>
__kernel__ stencil_iteration(...)
{
  // Initialization Instructions
  g_tx = ...;  g_ty = ...;
  ...
  __shared__ T shArr[3][sizey][sizex];
  first = 0; second = 1; third = 2;
  // Load first 2 planes
  shArr[0][][] = ;
  shArr[1][][] = ;
  for (k=halo_k; k<=zSize; k++) {
    // Load third plane to __shared__
    shArr[2][][] = ;
    __syncthreads();
    if (inside) {
     // stencil calculation
     ...
    }
    __syncthreads();
    // Shift planes
    first = (first+1)%3;
    second = (second+1)%3;
    third = (third+1)%3;
} }
```

(a) With Corner Accesses

```
#define sizey (BLOCK_Y+halo_j*2)
#define sizex (BLOCK_X+halo_i*2)

template <class T>
__kernel__ stencil_no_corner(...)
{
  // Initialization Instructions
  g_tx = ...;  g_ty = ...;
  ...
  __shared__ T shArr[sizey][sizex];

  // Load first 2 planes to registers
  T middle = ...;  T below = ...;
  for (k=halo_k; k<=zSize-halo_k; k++)
        {
    // Shift registers
    top = middle;
    middle = below;
    // load third plane to registers
    T below = ...;
    __syncthreads();
    // load middle plane to __shared__
    ...
    __syncthreads();
    if (inside) {
      // stencil calculation
      ...
} } }
```

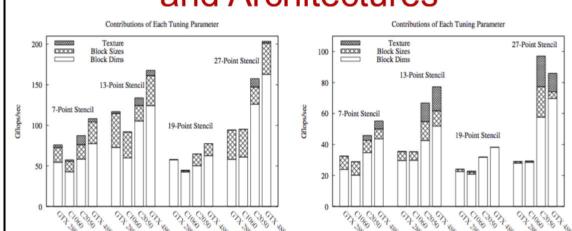
(b) Without Corner Accesses

Figure 5: Stencil Kernel Templates

Other Optimizations

- X dimension delivers coalesced global memory accesses
  - Pad to multiples of 32 stencil elements
- Halo regions are aligned to 128-bit boundaries
- Input (parameter) arrays are padded to match halo region, to share indexing.
- BlockSize.x is maximized to avoid non-coalesced accesses to halo region
- Blocks are square to reduce area of redundancy.
- Use of shared memory for input.
- Use of texture fetch for input.

Performance Across Stencils and Architectures



(a) Single Precision (SP) Stencils

(b) Double Precision (DP) Stencils

Figure 8: Stencil Tuning Effect Breakups

GPU Cluster Performance

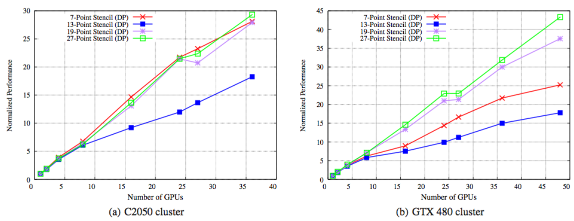


Figure 11: Weak Scaling of DP Stencils on GPU Clusters