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 $t$.

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++) {  
    // 3-d 7-point stencil
  }
  temp_ptr = A;
  A = B;
  B = temp_ptr;
}

Heat Equation, Add Tiling

for (kk=0; kk<timesteps; kk++) {  // 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
          }
        }
      }
    }
  }
  temp_ptr = A;
  A = B;
  B = temp_ptr;
}

Heat Equation, Time Skewing

for (kk=0; kk<nz-1; kk++) {  // time step loop
  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 i 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
            }
          }
        }
      }
    }
  }
  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

Parallel Stencils in Pochoir

![Image of heat equation, circular queue]![Image of heat equation, cache oblivious]![Image of pochoir stencil compiler specification]![Image of parallel stencils in pochoir]
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

- Explicitly limit performance and availability
- Ensure that blocking patterns exist, even in global-to
- Ensure that blocking for region blocks
- Keep padding, even in 3D

What about GPUs?

- Two recent papers:
  "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 \((n+2)\times(n+2)\) halo region is needed to compute an \(n\times n\) block if subscript expressions are of the form \(\pm 1\), for example
- By expanding the halo region, we can trade off redundant computation for reduced accesses to global memory when parallelizing across time steps.
2.5D Decomposition

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

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.
GPU Cluster Performance

Figure 11: Weak Scaling of DP Density on GPU Clusters