1D Stencil Example

February 25, 2009

1 Program

This example uses simple stencil computation (1D data, 2D iteration space) and shows series of function calls in the AlphaZ system to generate a parallel code. The example is taken (and simplified) from [Lamport]. The goal is to generate a program with outer sequential and inner parallel loops, so that the generated sequential code could be easily parallelized, for example, just by adding a pragma using OpenMP.

The alphabets program is available in 'examples/Stencil1D/OneDimSimple.ab'. Figure 1 is an (loosely) equivalent program in C. The inner loop updates 1D array B over some specified number of iterations TMAX. Because this is a Gauss-Seidel stencil computation, which updates a single array, the inner loop cannot be executed in parallel as it is. We describe a transformation to expose a parallel loop in the next section.

2 Schedule

The method to transform certain class of loops to a parallel loop is discussed formally in [Lamport]. Here we describe the steps using geometric intuition. Starting from the original iteration space in Figure 2, the red lines show sets of points that can be executed in parallel without violating the dependencies shown in black arrows. The red line can be viewed to be the vector $[-1, 2]$, and any vector $[-2, x]$, $2 \leq x < N$ satisfies all the dependencies, but it is obvious $[-1, 2]$ is most efficient (take the least number of lines to cover the iteration space).

The norm of vector $[-1, 2]$, $[2, 1]$ is a legal schedule for this program. Now we want this schedule to be aligned with one of the axes, so that a loop becomes the schedule loop (sequential), and the other loops can be declared parallel. There are two possible transformations to make the schedule aligned with one of the axes, one is to transform the program so that the red lines become horizontal, and the other is to make the lines vertical.

The transformation $T \begin{bmatrix} t \\ s \end{bmatrix}$ takes points in the original coordinate $(t, i)$ and maps to a new coordinate $(s, p)$ where $s$ is the sequential loop and $p$ is the parallel loop. $T = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}$ makes the red lines horizontal,

for $(t = 0; t < \text{TMAX}; t++)$
for $(i = 0; i <= \text{N}; i++)$
if $(t==0)$ {
} else if $(t > 0 \&\& (i == 0 || i == \text{N}))$
    $B[i] = B[i]$;
else {
    $B[i] = 0.5*(B[i-1] + B[i+1])$;
}
}

Figure 1: Original program in C loops.
resulting in an iteration space shown in Figure 3. This upper half of the transformation matrix $\begin{bmatrix} 2 & 1 \\ \end{bmatrix}$ comes from the scheduling vector we found earlier. The sequential schedule loop (s) in the new coordinates must use this schedule. The bottom half $\begin{bmatrix} 0 & 1 \\ \end{bmatrix}$ decides the orientation of the red line. $\begin{bmatrix} 0 & 1 \\ \end{bmatrix}$ means the $p$ coordinate is going to be the $i$ coordinate in the original loop, so that the red lines will be horizontal. If it were $\begin{bmatrix} 1 & 0 \\ \end{bmatrix}$, the $t$ coordinate in the original program becomes the $p$ coordinate, resulting in vertical red lines. The problem with this transformation is that because the transformation matrix is not uni-modal, the resulting iteration space has holes. Holes are integer points in the iteration space with no computation. The code generator cannot handle iteration space with holes, so this transformation cannot be used.

The other transformation that makes red lines vertical $\begin{bmatrix} 2 & 1 \\ 1 & 0 \\ \end{bmatrix}$, leads to iteration space shown in Figure 4. There are no holes in the resulting iterations space, and the code generator can handle this program. Since the schedule vector is aligned with $i$-axis, the $i$ loop in the resulting program becomes the time loop of the original stencil computation. The inner loop of the code generated after this transformation can be executed in parallel.

3 Memory

The transformation described above is enough to make a loop parallel, but there is an unsolved efficiency problem. Because alphabets programs are single assignment, the generated code requires $O(N^2 \cdot TMAX)$ memory. Because this is a stencil computation that iterates over 1D array, the memory requirement should be $O(N)$. The memory mapping of the original program is $\begin{bmatrix} 0 & 1 \\ \end{bmatrix} \begin{bmatrix} t \\ i \\ \end{bmatrix}$, since $t$ is the time loop and $i$ is the loop that iterates over 1D array. Ignore legality for the moment and suppose that you can use the original memory map after applying the transformation $\begin{bmatrix} 2 & 1 \\ 1 & 0 \\ \end{bmatrix}$. Then the new memory map from the $(s, p)$ coordinates are simply the composition of the original memory map and the inverse of the transformation. Thus, the new memory map would be $\begin{bmatrix} 0 & 1 \\ \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & -2 \\ \end{bmatrix} = \begin{bmatrix} 1 & -2 \end{bmatrix}$.

The legality can be observed visually. Similar to the relation ship between schedule vectors and red lines, the norm of the vector $\begin{bmatrix} 1 & -2 \end{bmatrix}$, $\begin{bmatrix} 2 & 1 \end{bmatrix}$ are the set of points that writes to the same memory location, shown as blue dashed line in Figure 5. The figure shows a memory location is only written every other schedule time step, and the dependencies show that all uses of a variable uses values from the immediately
Figure 3: Program transformed so that the schedule is the vertical vector. Notice the holes in iteration space caused by the transformation not being uni-modular.

Figure 4: Program after transformation to make the schedule horizontal vector. Now there are no holes in iteration space.
Figure 5: Efficient memory map after transformation. All points on a blue dashed line writes to the same memory location.

preceeding time step. Now we have a program that only use O(N) memory.

4 AlphaZ commands

The following sequence of commands for the bean shell interface performs all of the above steps within the AlphaZ system and produces the code. (Assuming you are in examples/Stencil1D)

```
ConnectServer("alphabets.corequations.com");
ReadProgram("OneDimSimple.ab");
CoB("OneDTwoD", "B", "(t,i->2t+i,t)");
CoB("OneDTwoD", "Local", "(t,i->2t+i,t)");
SetMemorySpec("OneDTwoD", "Local", "(s,p->s-2p)");
GenerateFixTiledC("1,1"); GenerateWrapper();
GenerateWrapper();
```

References