Compiler/Run-Time Framework for Dynamic Data-Flow Parallelization of Tiled Programs

Martin Kong¹ Antoniu Pop² R. Govindarajan³
Louis-Noël Pouchet¹ Albert Cohen⁴ P. Sadayappan¹

¹ The Ohio State University
² The University of Manchester
³ Indian Institute of Science
⁴ Inria

January 19th, 2015
IMPACT 2015
5th International Workshop on Polyhedral Compilation Techniques
Amsterdam, Netherlands
Motivating Example: Blur-Roberts

Focus of this work: removal of data-parallel barriers executed on shared-memory multi-core machines

```c
for (i = 1; i < N - 1; i++)
    for (j = 1; j < N - 1; j++)
                      A[i][j+1] + A[i+1][j] +
                      A[i-1][j] + A[i-1][j-1] +
                      A[i-1][j+1] + A[i+1][j-1] +
                      A[i+1][j+1])/8.0;
```

```c
for (i = 1; i < N-2; i++)
    for (j = 2; j < N-1; j++)
        S2: A[i][j] = abs(B[i][j]-B[i+1][j-1]) +
                       abs(B[i+1][j] - B[i][j-1]);
```

- Barrier involves global consensus
- Number of synchronizations depend upon: program structure and applied transformations
- Some transformations could derive on loss of locality or parallelism

Blur-Roberts kernel performance in GFLOPS/sec for AMD Opteron 6274 (16 cores) and Intel Xeon E5-2650 (8 cores), on 1, half and all cores.
Tiled Blur-Roberts

Blur-Roberts tiled with PLuTo using the Minfuse heuristic (maximal decomposition)

if (_PB_N >= 3) {
  lbp=0;
  ubp=floord(_PB_N-2,32);
}

#pragma omp parallel for private(lbv,ubv)
for (t2=lbp;t2<=ubp;t2++) {
  for (t3=0;t3<=floord(_PB_N-2,32);t3++)
    for (t4=max(1,32*t2);t4<=min(_PB_N-2,32*t2+31);t4++) {
      lbv=max(1,32*t3);
      ubv=min(_PB_N-2,32*t3+31);
      //pragma ivdep
      //pragma vector always
      for (t7=lbv;t7<=ubv;t7++)
        B[t4][t7] = (A[t4][t7] + A[t4][t7-1] + A[t4][t7+1]) + A[t4][t7+1];
      }
    }
  }
}

Good parallelism, good vectorization!
Bad locality!
Two barriers

OSU / UM / IISC / Inria

Blur-Roberts tiled with PLuTo using the Smartfuse heuristic (fuse matching dimensions)

for (t1=0;t1<=floord(_PB_N-2,16);t1++) {
  lbp=max(0,ceil(t2*_PB_N-2,32));
  ubp=min(floord(_PB_N-3,32));
   #pragma omp parallel for private(lbv,ubv)
for (t2=lbp;t2<=ubp;t2++) {
  if (t2 == 0) && (lvp == ceil(lvp,_PB_N-2,32))
    for (t4=max(1,32*t2);t4<=min(_PB_N-2,32*t2+31);t4++)
      B[t4][t7] = (A[t4][t7] + A[t4][t7-1] + A[t4][t7+1]) + A[t4][t7+1];
  else
    for (t4=max(1,32*t2);t4<=min(_PB_N-2,32*t2+31);t4++)
      B[t4][t7] = (A[t4][t7] + A[t4][t7-1] + A[t4][t7+1]) + A[t4][t7+1];
}
}

Good locality!
"Bad" parallelism, poor vectorization!
One barrier executed $O(n)$ times!
Our solution

- Decompose
- Tile to coarsen granularity

Apply PluTo tiling algorithm

- Extract tile-level polyhedral abstractions

- Compute PRDG from tile-level abstractions

- Build task graph from decorated PRDG and collect input/output dependence info

- Partition tile-level domains by dependence signatures

- Prune duplicated tile dependences
- Prune non-forward tile dependences

- Keep partitions separated
- Replicate internal structures
- Generate stream declarations
- Pragmatization (clause generation from dependence signature)

- Project schedule onto selected tile dimensions
- Compute tile domains
- Compute tile dependences

- Expand PRDG nodes to accommodate partitions
- Remap PRDG edges according to the dependences signatures

- Prune covering dependences
- Prune by transitive reduction
- Compute stream sizes

OSU / UM / IISC / Inria