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

Martin Kong[1] Antoniu Pop[2] R. Govindarajan[3]
Louis-Noël Pouchet[1] Albert Cohen[4] P. Sadayappan[1]

[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

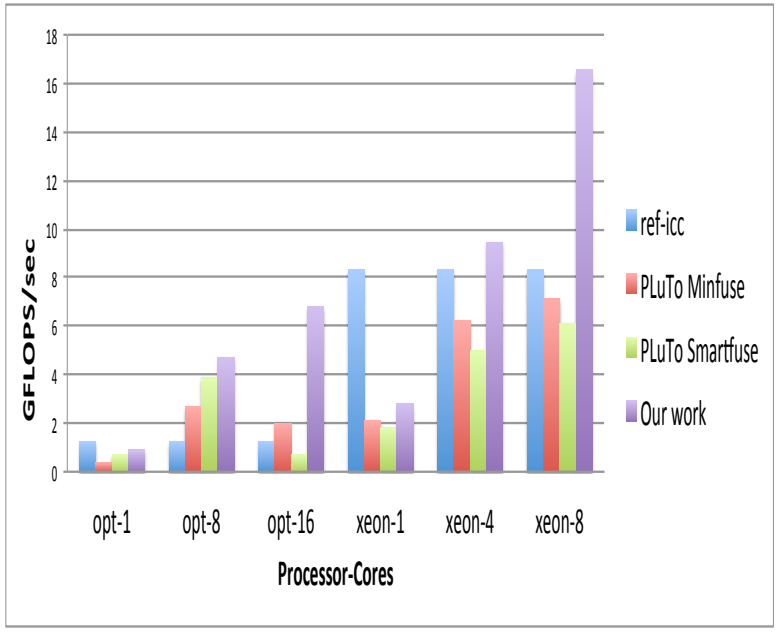
Motivating Example: Blur-Roberts

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

```
for (i = 1; i < N - 1; i++)
  for (j = 1; j < N - 1; j++)
S1: B[i][j] = (A[i][j] + A[i][j-1] +
      A[i][1+j] + A[1+i][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;

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 involve 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][1+t7] + A[1+t4][t7] + A[t4-1][t7] +
                       A[t4-1][t7-1] + A[t4-1][t7+1] + A[t4+1][t7-1] + A[t4+1][t7+1])/8.0;;
      }
}
if (_PB_N >= 4) {
    lbp=0;
    ubp=floord(_PB_N-3,32);
#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-3,32*t2+31);t4++) {
          lbv=max(2,32*t3);
          ubv=min(_PB_N-2,32*t3+31);
          #pragma ivdep
          #pragma vector always
          for (t7=lbv;t7<=ubv;t7++)
            A[t4][t7] = (B[t4][t7]-B[t4+1][t7-1]) + (B[t4+1][t7] - B[t4][t7-1]);;
        }
}
```

Good parallelism, good vectorization!
Bad locality!
Two barriers

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,ceild(32*t1-_PB_N+2,32));
  ubp=min(floord(_PB_N-1,32),t1);
#pragma omp parallel for private(lbv,ubv)
  for (t2=lbp;t2<=ubp;t2++) {
    if ((t1 == t2) && (t1 <= floord(_PB_N-2,32)))
      for (t4=max(1,32*t1);t4<=min(_PB_N-2,32*t1+31);t4++)
        B[1][t4] = (A[1][t4] + A[1][t4-1] + A[1][1+t4] + A[1+1][t4] + A[1 -1][t4] + A[1 -1][t4-1] + A[1 -1][t4+1] + A[1 +1][t4-1] + A[1 +1][t4+1])/8.0;;
    if (32*t2 == _PB_N-1)
      for (t3=max(2,32*t1-_PB_N+1);t3<=32*t1-_PB_N+32;t3++)
        if ((_PB_N+31)%32 == 0)
          A[t3-1][_PB_N-2] = (B[t3-1][_PB_N-2]-B[t3-1 +1][_PB_N-2 -1]) + (B[t3-1 +1][_PB_N-2] - B[t3-1][_PB_N-2 -1]);;
    if ((_PB_N >= 5) && (_PB_N <= 32) && (t1 == 0) && (t2 == 0)) {
      for (t3=2;t3<=_PB_N-2;t3++) {
        for (t4=1;t4<=2;t4++) {
          B[t3][t4] = (A[t3][t4] + A[t3][t4-1] + A[t3][1+t4] + A[1+t3][t4] + A[t3-1][t4] + A[t3-1][t4-1] + A[t3-1][t4+1] + A[t3+1][t4-1] + A[t3+1][t4+1])/8.0;;
        }
        for (t4=3;t4<=_PB_N-2;t4++) {
          B[t3][t4] = (A[t3][t4] + A[t3][t4-1] + A[t3][1+t4] + A[1+t3][t4] + A[t3-1][t4] + A[t3-1][t4-1] + A[t3-1][t4+1] + A[t3+1][t4-1] + A[t3+1][t4+1])/8.0;;
          A[t3-1][t4-1] = (B[t3-1][t4-1]-B[t3-1 +1][t4-1 -1]) + (B[t3-1 +1][t4-1] - B[t3-1][t4-1 -1]);;
        }
        A[t3-1][_PB_N-2] = (B[t3-1][_PB_N-2]-B[t3-1 +1][_PB_N-2 -1]) + (B[t3-1 +1][_PB_N-2] - B[t3-1][_PB_N-2 -1]);;
      }
    }
    if ((_PB_N >= 33) && (t2 == 0)) {
      for (t3=max(2,32*t1);t3<=min(_PB_N-2,32*t1+31);t3++) {
        for (t4=1;t4<=2;t4++)
          B[t3][t4] = (A[t3][t4] + A[t3][t4-1] + A[t3][1+t4] + A[1+t3][t4] + A[t3-1][t4] + A[t3-1][t4-1] + A[t3-1][t4+1] + A[t3+1][t4-1] + A[t3+1][t4+1])/8.0;;
        for (t4=3;t4<=31;t4++) {
          B[t3][t4] = (A[t3][t4] + A[t3][t4-1] + A[t3][1+t4] + A[1+t3][t4] + A[t3-1][t4] + A[t3-1][t4-1] + A[t3-1][t4+1] + A[t3+1][t4-1] + A[t3+1][t4+1])/8.0;;
          A[t3-1][t4-1] = (B[t3-1][t4-1]-B[t3-1 +1][t4-1 -1]) + (B[t3-1 +1][t4-1] - B[t3-1][t4-1 -1]);;
        }
      }
    }
    if ((_PB_N == 4) && (t1 == 0) && (t2 == 0)) {
      for (t4=1;t4<=2;t4++)
        B[2][t4] = (A[2][t4] + A[2][t4-1] + A[2][1+t4] + A[1+2][t4] + A[2 -1][t4] + A[2 -1][t4-1] + A[2 -1][t4+1] + A[2 +1][t4-1] + A[2 +1][t4+1])/8.0;;
      A[1][2] = (B[1][2]-B[1 +1][2 -1]) + (B[1 +1][2] - B[1][2 -1]);;
    }
    if ((t2 <= floord(_PB_N-2,32)) && (t2 >= ceild(_PB_N-32,32)))
      for (t3=max(2,32*t1-32*t2);t3<=min(min(_PB_N-2,32*t1-1),32*t1-32*t2+31);t3++) {
        for (t4=32*t2;t4<=_PB_N-2;t4++) {
          B[t3][t4] = (A[t3][t4] + A[t3][t4-1] + A[t3][1+t4] + A[1+t3][t4] + A[t3-1][t4] + A[t3-1][t4-1] + A[t3-1][t4+1] + A[t3+1][t4-1] + A[t3+1][t4+1])/8.0;;
          A[t3-1][t4-1] = (B[t3-1][t4-1]-B[t3-1 +1][t4-1 -1]) + (B[t3-1 +1][t4-1] - B[t3-1][t4-1 -1]);;
        }
        A[t3-1][_PB_N-2] = (B[t3-1][_PB_N-2]-B[t3-1 +1][_PB_N-2 -1]) + (B[t3-1 +1][_PB_N-2] - B[t3-1][_PB_N-2 -1]);;
      }
    if (t2 <= floord(_PB_N-33,32))
      for (t3=max(2,32*t1-32*t2);t3<=min(min(_PB_N-2,32*t1-1),32*t1-32*t2+31);t3++)
        for (t4=32*t2;t4<=32*t2+31;t4++) {
          B[t3][t4] = (A[t3][t4] + A[t3][t4-1] + A[t3][1+t4] + A[1+t3][t4] + A[t3-1][t4] + A[t3-1][t4-1] + A[t3-1][t4+1] + A[t3+1][t4-1] + A[t3+1][t4+1])/8.0;;
          A[t3-1][t4-1] = (B[t3-1][t4-1]-B[t3-1 +1][t4-1 -1]) + (B[t3-1 +1][t4-1] - B[t3-1][t4-1 -1]);;
        }
  }
}
```

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

Our solution