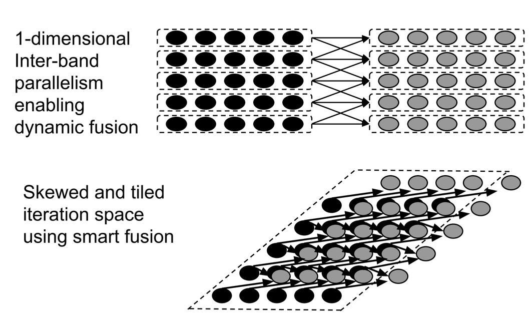
Compiler/Run-Time Framework for Dynamic Data-Flow Parallelization of Tiled Programs

HIPEAC'15: 10th International Conference on High-Performance Embedded Architectures and Compilers

Problem

- Over-synchronization: fixed or problem size and transformation dependant
- Trade-off between locality and parallelism
- Poor load balance (e.g. wavefront technique)
- Extracting coarse-grained parallelism could:
  - Augment code complexity (long codes, hard to vectorize)
  - Impact negatively the intra-tile performance (e.g. deeply nested modulo conditions)
  - Outer parallel loops could still be deeply nested

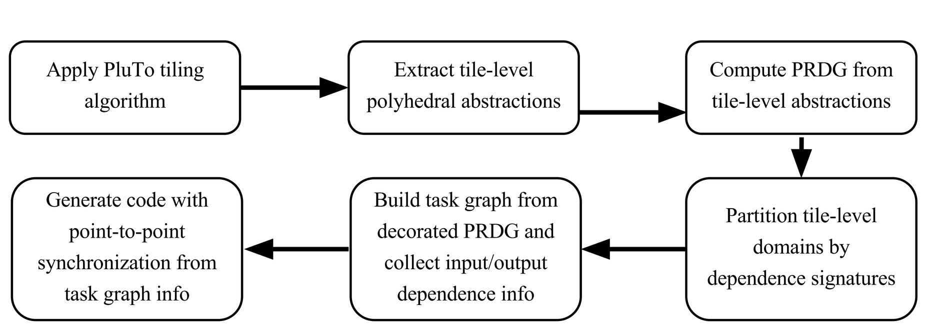
```
for (i = 1; i < N - 1; i++)
  for (j = 1; j < N - 1; j++)
S1: B[i][j] = (A[i][j] + A[i][j-1] +
      A[i][1+j] + A[1+i][j] +
      A[i-1][j] + A[i-1][j-1] +
      A[i-1][j+1] + A[i+1][j-1] +
      A[i+1][j+1])/8.0;

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]);
```



Contributions

- Novel technique for removing barriers:
  - Operate on the tiled space (used to coarsen) before code generation stage
  - Partitioning: produces (tiled) domains with unique dependence signature
- Two barrier removal flavors:
  - Inter-band
  - Intra-band (aka. dynamic wavefront)
- Analyses to statically prune dependences: alleviates runtime burden
- Specific code generation step
  - Keep separated the partitions
  - Compute stream sizes
  - Generate stream declarations
  - Pragmatization (clause generation from dependence signature)

High-level Flow

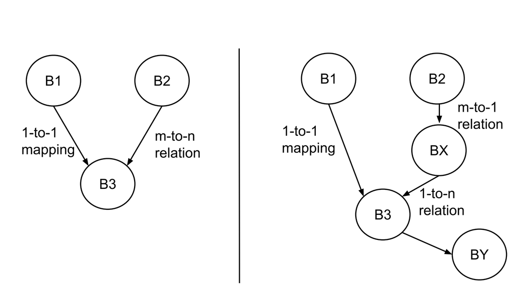


Inter-band Parallelism

Definition 1 (Inter-band parallelism) *Given two distinct bands A and B. Barrier-less inter-band parallelism is exploitable if:*

- *there exists at least one point in band B that does not depend on all the points of band A*
- *Neither band A nor band B have dependence cycles*

```
for (i=0; i<l; i++) // <-- Band 1
  for (j=0; j<m; j++)
    for (k=0; k<q; k++)
 S1:  A[i][j] += B[i][k] * C[k][j];
for (i=0; i<m; i++) // <-- Band 2
  for (j=0; j<n; j++)
    for (k=0; k<p; k++)
 S2:  D[i][j] += E[i][k] * F[k][j];
for (i=0; i<l; i++) // <-- Band3
  for (j=0; j<n; j++)
    for (k=0; k<m; k++)
 S3:  G[i][j] += A[i][k] * D[k][j];
```



References

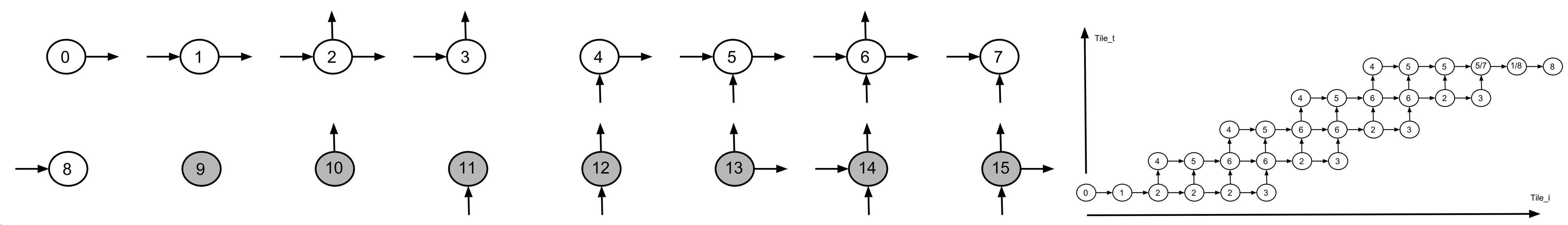
[1] Pop, Antoniu and Cohen, Albert. OpenStream: Expressiveness and data-flow compilation of OpenMP streaming programs. In ACM Transactions on Architecture and Code Optimization (TACO), volume 9, 2013

[2] Verdoolaege, Sven and Carlos Juega, Juan and Cohen, Albert and Ignacio Gómez, José and Tenllado, Christian and Catthoor, Francky. Polyhedral Parallel Code Generation for CUDA. In ACM Transactions on Architecture and Code Optimization (TACO), volume 9, 2013

Enabling Data-Flow Parallelism

Definition 2 (Dependence Signature) *The dependence signature* SIG^S *of a domain* I^S_{tile} *is composed of two sets: the IN set and the OUT set. For each dependence relation* k, k *is put in IN (resp. OUT) iff* $\mathcal{D}^{T\to S}$ *(resp.* $\mathcal{D}^{S\to T}$*) has at least one destination (resp. source) in* I^S_{tile}.

$$\text{SIG}^S = \{IN^S, OUT^S\},\ \ IN^S = \{k : Ran(D^{k:T\to S}_{tile}) \cap I^S_{tile} \neq \varnothing\},$$
$$OUT^S = \{k : Dom(D^{k:S\to T}_{tile}) \cap I^S_{tile} \neq \varnothing\}$$



Example

```
int band_stream_ii_size = (floor((15 + ni)/16));
int band_stream_ii[band_stream_ii_size] __attribute__
((stream));
int read_window[W];
int write_window[W];

for (int ii = 0; ii <= floord(ni - 1, 16); ii += 1)
#pragma omp task  output(band_stream_ii[ii] << write_window[W])
 for (int jj = 0; jj <= floord(nj - 1, 16); jj += 1)
  for (int i = 16 * ii; i <= min(ni - 1, 16 * ii + 15); i++)
   for (int j = 16 * jj; j <= min(nj - 1, 16 * jj + 15); j++)
    C[i][j] *= beta;

for (int ii = 0; ii <= floord(ni - 1, 16); ii++)
#pragma omp task  input(band_stream_ii[ii] >> read_window[W])
 for (int jj = 0; jj <= floord(nj - 1, 16); jj++)
  for (int kk = 0; kk <= floord(nk - 1, 16); kk++)
   for (int i = 16 * ii; i <= min(ni - 1, 16 * ii + 15); i++)
    for (int j = 16 * jj; j <= min(nj-1, 16 * jj + 15); j++)
     for (int k = 16 * kk; k <= min(nk-1, 16 * kk + 15); k++)
      C[i][j] += ((alpha * A[i][k]) * B[k][j]);
```

1-to-1 mapping

Tiled DGEMM parallelized with OpenStream

```
for (int tt = 1; tt < (tsteps-1) / 16; tt++)
 for (int ii = 2*tt+2; ii <= 2*tt + (n-3) / 16; ii++)
  #pragma omp task \
    input(\
      stream_tt[(tt) * S_1_3 + ii] >> token1_1,\
      stream_ii[(tt) * S_1_3 + ii] >> token2_1\
    )\
    output(\
      stream_tt[(tt+1) * S_1_3 + ii] << token1_2,\
      stream_ii[(tt) * S_1_3 + 1+ii] << token2_2\
    )
  for (int jj=ii; jj <= ii+(n-3)/16+1; jj++)
   for (int t=max(16*tt,-n+8*jj+2); t <= 16*tt+15; t++)
    for (int i=max(16*ii,-n+16*jj+2);
         i<=min(min(16*ii+15,16*jj+14),n+2*t-1); i++)
     for (int j=max(i+1,16*jj); j <= min(16*jj+15,n+i-2); j++) {
      if (n+2*t >= i+2)
        B[-2*t+i][-i+j] = 0.2 * ( A[-2*t+i][-i+j] +
          A[-2*t+i][-i+j-1] + A[-2*t+i][-i+j+1] +
          A[-2*t+i+1][-i+j] + A[-2*t+i-1][-i+j] );
      A[-2*t+i-1][-i+j] = B[-2*t+i-1][-i+j];
     }
```

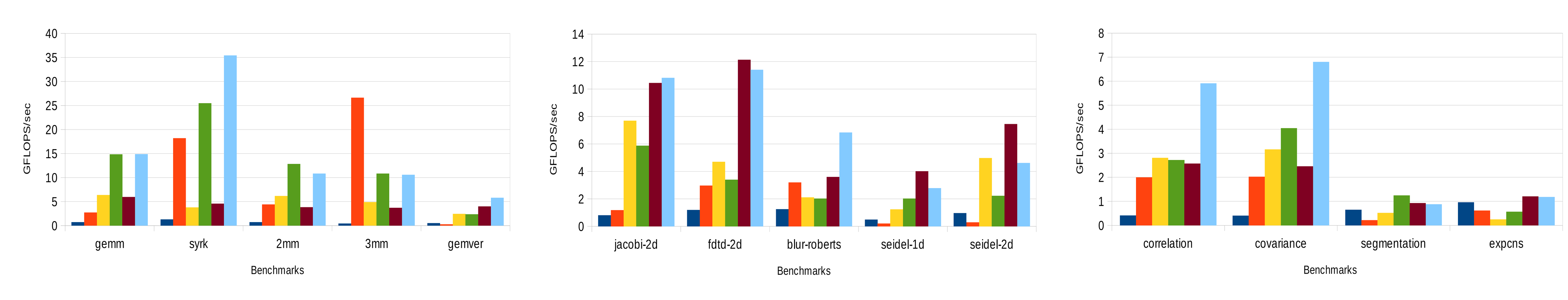
Indexing Tile Coordinates

Partition Dependence Signature

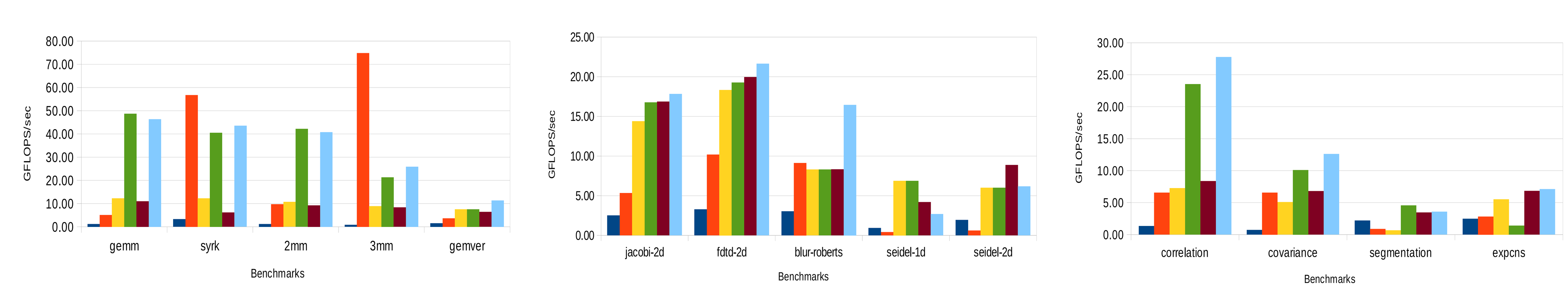
Task Body

Tiled and partitioned Jacobi-2d - steady state partition

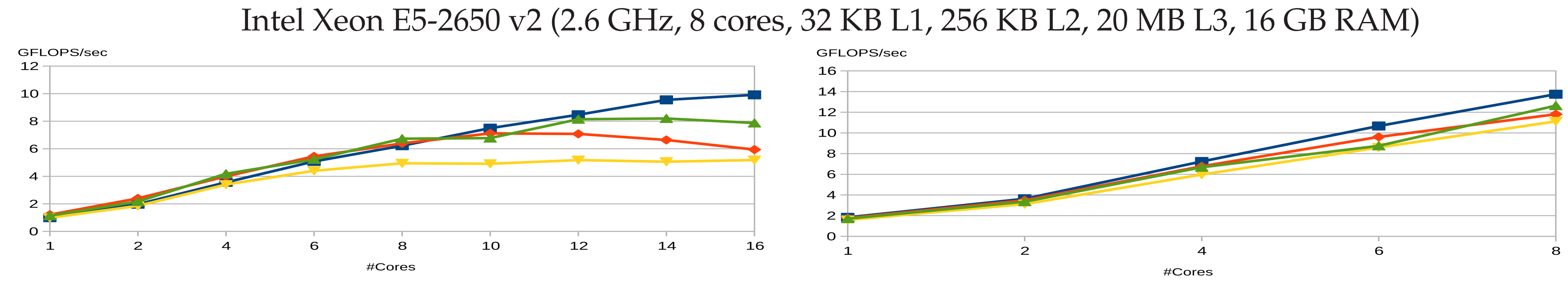
Results



AMD Opteron 6274 (2.2 GHz, 16 cores, 16 KB L1, 8 x 2 MB L2, 6 MB L3, 32 GB RAM)



Intel Xeon E5-2650 v2 (2.6 GHz, 8 cores, 32 KB L1, 256 KB L2, 20 MB L3, 16 GB RAM)



Dynamic Wavefront vs Diamond Tiling: AMD Opteron 6274 (left) and Intel Xeon E5-2650 v2 (right)

Authors

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