Automating Scheduling

Logistics
- Final report for project due this Friday, 5/4/12
- Quiz 4 due this Monday, 5/7/12
- Poster session Thursday May 10 from 2-4pm
  - Distance students need to contact me to set up a skype time ASAP

Today
- Automating scheduling with the Farkas lemma (missed a step)
- Start Sparse Polyhedral Framework (SPF) for run-time reordering transformations

Fourier-Motzkin and Farkas Questions (HW10)

1D scheduling
- Specify the data dependence relation with all inequality constraints. The result is a data dependence polyhedron.
- Want the function $\theta_S(j) - \theta_R(i) - 1$ to be non-negative over the dependence polyhedron
- Use Farkas lemma to set up a new set of constraints

\[
\begin{align*}
\theta_R(i) &= \bar{v}^T \bar{i} + \bar{b} \\
\theta_S(j) &= \bar{w}^T \bar{j} + \bar{c}
\end{align*}
\]

\[\theta_S(j) - \theta_R(i) - 1 = \lambda_0 + \bar{\lambda}^T (A \begin{bmatrix} \bar{i} \\ \bar{j} \\ \bar{p} \\ 1 \end{bmatrix})\]

- Collect coefficients for each term on both sides of equality to create a set of affine equalities involving $v$, $w$, $b$, $c$, and $\lambda$s
- Use Fourier-Motzkin to project out all variables except for $v$, $w$, $b$, and $c$
The process of determining set of legal schedules

(1) Change all of the equality constraints in $D_{R \rightarrow S}$ to inequality constraints.

$$D_{R \rightarrow S} = \{ [\vec{i} \rightarrow \vec{j} | A'] \vec{i} \vec{j} \geq \vec{0} \}$$

(2) Use the Farkas lemma to create a set of constraints for the schedule.

$$\theta_S(\vec{j}) - \theta_R(\vec{i}) - 1 = \lambda_0 + \vec{\lambda}^T (A \begin{bmatrix} \vec{i} \\ \vec{j} \\ \vec{p} \\ 1 \end{bmatrix})$$

$$\lambda_0 \geq 0 \text{ and } \vec{\lambda} \geq \vec{0}$$

$$\theta_R(\vec{i}) = \vec{\sigma}^T \vec{i} + \vec{b}$$

$$\theta_S(\vec{j}) = \vec{\omega}^T \vec{j} + \vec{c}$$

(3) Collect coefficients for each term to create set of equalities.

(4) Solve for $\vec{v}$, $\vec{w}$, $\vec{b}$, and $\vec{c}$ vector constraints by projecting out lambdas.

Example of using the Farkas lemma

Original code (problem from HW10)

```
do i = 0,N-1
  do j = 0,N-1
    A(i,j) = A(i-1,j-1)*.05
  enddo
enddo
```

(1) Dependence polyhedron

$$D_{i \rightarrow j} = \{ [i_1, j_1] \rightarrow [j_2, j_2] | [1 0 -1 0 0 1 -1 0 1 0 0 -1 0 1 0 -1] \geq 0 \}$$

(2) Farkas lemma to set up constraints

(3) Collect coefficients for each term to create set of equalities

(4) Project out lambdas to determine set of legal schedules
HW10 problem continued

\[ \theta(i_2,j_2) - \theta(i_1,j_1) - 1 = \lambda_0 + \tilde{\lambda}^T A \]

(2) Farkas lemma to set up constraints

\[
\begin{align*}
\lambda_1 & \quad 1 \quad 0 \quad -1 \quad 0 \quad 0 \quad 1 \\
\lambda_2 & \quad -1 \quad 0 \quad 1 \quad 0 \quad 0 \quad -1 \\
\lambda_3 & \quad 0 \quad 1 \quad 0 \quad -1 \quad 0 \quad 1 \\
\lambda_4 & \quad 0 \quad -1 \quad 0 \quad 1 \quad 0 \quad -1 \\
\lambda_5 & \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \\
\lambda_6 & \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \\
\lambda_7 & \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \\
\lambda_8 & \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \\
\lambda_9 & \quad -1 \quad 0 \quad 0 \quad 0 \quad 1 \quad -1 \\
\lambda_{10} & \quad 0 \quad -1 \quad 0 \quad 0 \quad 1 \quad -1 \\
\lambda_{11} & \quad 0 \quad 0 \quad -1 \quad 0 \quad 1 \quad -1 \\
\lambda_{12} & \quad 0 \quad 0 \quad 0 \quad -1 \quad 1 \quad -1 \\
\end{align*}
\]

(3) Collect coefficients for each term to create set of equalities

\[
\begin{align*}
-a & = \lambda_1 - \lambda_2 + \lambda_6 - \lambda_9 \\
-b & = \lambda_3 - \lambda_4 + \lambda_6 - \lambda_{10} \\
a & = -\lambda_3 - 3 + \lambda_7 - \lambda_{11} \\
b & = -\lambda_3 - \lambda_4 + \lambda_9 - \lambda_{12} \\
0 & = \lambda_6 + \lambda_{10} + \lambda_{11} + \lambda_{12} \\
-1 & = \lambda_6 + \lambda_3 - \lambda_2 + \lambda_5 - \lambda_4 - \lambda_9 - \lambda_{10} - \lambda_{11} - \lambda_{12} \\
\lambda_x & \geq 0, \forall 0 \leq x \leq 12
\end{align*}
\]

(4) Project out lambdas to determine set of legal schedules

\[ \{[a,b,c]: 1 <= a+b\} \]
Automating Run-Time Reordering Transformations with the Sparse Polyhedral Framework (SPF) and Arbitrary Task Graphs

Michelle Mills Strout
CS560 May 1, 2012
Somewhat modified from Imperial College talk given November 21, 2011

The Big Picture Problem

- Data movement is expensive …
  - in terms of execution time
  - in terms of power usage
- Data reordering and/or loop transformations can turn data reuse into data locality
- Research in the polyhedral model has led to significant automation for loop transformations that affect data locality
- However, sparse/irregular computations do not fit in the polyhedral model
Goal: Turn Data Reuse into Data Locality

- **Spatial locality** occurs when memory locations mapped to the same cache-line are used before the cache line is evicted.
- **Temporal locality** occurs when the same memory location is reused before its cache line is evicted.

![Diagram](image)

**Colorado State University**

---

***Run-time Reordering Transformations***

**Inspector**
- Traverses index array
- Generates data reordering function $\sigma$
- Reorder data and updates index array

**Original Code**

```
for i=0,7
  Y[i] = Z[r[i]]
```

**Executor**

```
for i=0,7
  Y[i] = Z'[r'[i]]
```

---

**Generates data reordering function $\sigma$**

```
for i=0,7
  ... r[i] ...
for j=0,7
  sigma[j] = ...
for j=0,7
  Z'[sigma[j]] = Z[j]
  r'[j] = sigma[r[j]]
```
Example Inspector/Executor Strategies

- Gather/scatter parallelization [Saltz et al. 94]
- Cache blocking [Im & Yelick 98]
- Irregular cache blocking [Douglas & Rude 00]
- Full sparse tiling (ICCS 2001)
- Communication avoiding [Demmel et al. 08]
- Run-time data and iteration permutation [Chen and Kennedy 99, Mitchell 99, …]
- Compositions of the above (PLDI 2003)

Inspector/Executor Strategies show great promise BUT…

- Only a couple have been automated
- There is library support for some I/E strategies, but specializing the library for the given sparse data structures is non-trivial
- How can we automate or semi-automate the application of I/E strategies?
Run-time Reordering Transformations

- Challenge: unable to effectively reorder data and computation at compile-time in irregular applications
- Approach: run-time reordering transformations
- Vision:

  Transformation Framework
  SPF
  SPF
  SPF

  Inspector/Executor Code
  Decision Code
  Compiler
  Run-time

Sparse Polyhedral Framework (SPF)

- Adds uninterpreted functions to the polyhedral framework
  - Polyhedral model includes affine inequality constraints to represent iteration spaces.
  - SPF adds constraints such as $x=f(y)$, where $f$ is a function and its input domain and output range are polyhedra.
  - [Pugh & Wonnacott 94] used for data dependence analysis. SPF uses to represent transformations.
- Code generation for SPF results in inspector and executor code.
Run-time Reordering Transformations

Traverses index array
Generates data reordering function $\sigma$
Reorders data and updates index array

**Inspector**

```
for i=0,7
  ... r[i] ...
for j=0,7
  sigma[j] = ...
for j=0,7
  Z'[sigma[j]]=Z[j]
  r'[j]=sigma[r[j]]
```

**Executor**

```
for i=0,7
  Y[i] = Z'[r'[i]]
```

Original Code

```
for i=0,7
  Y[i] = Z[r[i]]
```

Computation Specification in SPF

**Original Code**

```
for i=0,7
  Y[i] = Z[r[i]]
```

- Each data array has a data space
  $Y_0 = \{[y] | 0 \leq y \leq 7\}$  $Z_0 = \{[z] | 0 \leq z \leq 7\}$
- Each index array is represented with and uninterpreted function $r()$
  and has a domain and range
  $\{[v] \rightarrow [w] | 0 \leq v, w \leq 7\}$
Computation Specification in SPF cont …

**Original Code**

```
for i=0,7
   Y[i] = Z[r[i]]
```

- Each statement represented with …
  - An iteration space set $S_0 = \{[i] | 0 \leq i \leq 7\}$
  - Scheduling function $S_{S_0 \rightarrow I_0} = \{[i] \rightarrow [0, i, 0]\}$
  - Access functions for each data array reference
    
    $A_{S_0 \rightarrow Y_0} = \{[i] \rightarrow [i]\}$
    
    $A_{S_0 \rightarrow Z_0} = \{[i] \rightarrow [z] | z = r(i)\}$

IEGenCC tool helps generate the SPF specification of algorithm.
Transformation Specification in SPF and Inspector/Executor Generator (IEGen)

Data Reordering Transformation Relation

\[ R_{Z_0 \rightarrow Z_1} = \{ [z] \rightarrow [z'] \mid z' = \sigma(z) \} \]

Inspector Dependence Graph (IDG)

Executor

for \( i = 0, 7 \)
... \( r[i] \) ...
for \( j = 0, 7 \)
\( \sigma[j] = \) ...
for \( j = 0, 7 \)

\( Z'[\sigma[j]] = Z[j] \)
\( r'[j] = \sigma[r[j]] \)

Each statement represented with ...

- An iteration space set \( S_0 = \{ [i] \mid 0 \leq i \leq 7 \} \)
- Scheduling function \( S_{S_0 \rightarrow I_0} = \{ [i] \rightarrow [0, i, 0] \} \)
- Access functions for each array reference

\[ A_{S_0 \rightarrow Y_0} = \{ [i] \rightarrow [i] \} \]

\[ A_{S_0 \rightarrow Z_1} = \{ [i] \rightarrow [z'] \mid z' = \sigma(r(i)) = r'(i) \} \]
Implementation Details: Transformations on Computation and Data Spaces

Reordering transformations modify the computation specification

Data Reordering followed by Pointer Update

\[ R_{Z_0 \rightarrow Z_1} = \{ [z] \rightarrow [z'] \mid z' = \sigma(z) \} \]

\[ r'(v) = \{ [v] \rightarrow [w] \mid w = \sigma(r(v)) \land 0 \leq v, w \leq 7 \} \]

Inspector and Executor Code Gen

Inspector generated from Inspector Dependence Graph created during transformation process

Executor generated from transformed algorithm specification
Computation and Transformation Specification in SPF

- Data and index array specifications
- Each statement represented with …
  - An iteration space set
  - A schedule mapping to full iteration space
  - Access functions for each data array reference
- Data dependences relations between iterations in full iteration space
- Transformation specification is a sequence of data and iteration reorderings represented as integer tuple relations

Key Insights in SPF and IEGen

- The inspectors traverse the access relations and/or the data dependences
- We can express how the access relations and data dependences will change
- Subsequent inspectors traverse the new data mappings and data dependences
- Use polyhedral code generator (Cloog) for outer loops and deal with sparsity in inner loops and access relations
Another Example (MOLDYN)

For $s=1,T$
  For $i=1,n$
    $... = ... Z[i]$  
  Endfor

  For $j=1,m$
    $Z[l[j]] = ...$
    $Z[r[j]] = ...$
  Endfor

  For $k=1,n$
    $Z[k] += ...$
  Endfor

Access Relation for i loop
$A_{I_0 \rightarrow Z_0} = \{[i] \rightarrow [i]\}$

Access Relation for j loop
$A_{J_0 \rightarrow Z_0} = \{[j] \rightarrow [i]| i = l(j) \lor i = r(j)\}$

Data Dependences between i and j loop
$D_{I_0 \rightarrow J_0} = \{[i] \rightarrow [j]| (i = l(j)) \lor (i = r(j))\}$

Data Permutation Reordering
(Equations are the compile-time abstraction)

$R_{Z_0 \rightarrow Z_1} = T_{I_0 \rightarrow I_1} = \{[i] \rightarrow [\sigma(i)]\}$

CPACK reordering heuristic [Ding & Kennedy 99]

$A_{J_0 \rightarrow Z_0} = \{[j] \rightarrow [i]| i = l(j) \lor i = r(j)\}$

$A_{J_0 \rightarrow Z_1} = \{[j] \rightarrow [i]| i = \sigma(l(j)) \lor i = \sigma(r(j))\}$
Effect of Data Reordering on Inspector Dependence Graph (IDG)

$$T_{J_0 \rightarrow J_1} = \{ [j] \rightarrow [x] \mid x = \delta(j) \}$$

$$A_{J_0 \rightarrow Z_1} = \{ [j] \rightarrow [i] \mid i = \sigma(l(j)) \lor i = \sigma(r(j)) \}$$

$$A_{J_1 \rightarrow Z_1} = \{ [j] \rightarrow [i] \mid i = \sigma(l(\delta^{-1}(j))) \lor i = \sigma(r(\delta^{-1}(j))) \}$$

Iteration Permutation Reordering
IDG After Iteration Permutation

Dependences Between Loops after other transformations

for s=1,T
  for i=1,n
    ... = ...Z[i]
  endfor
  for j=1,m
    Z[l[j]] = ...
    Z[r[j]] = ...
  endfor
  for k=1,n
    ... += Z[k]
  endfor
endfor

\[ D_{I_1 \rightarrow J_1} = \{ [0, i] \rightarrow [1, j] \mid i = \sigma(l(\delta^{-1}(j))) \wedge i = \sigma(r(\delta^{-1}(j))) \} \]
Full Sparse Tiling (FST)

\[ T_{F_1 \rightarrow F_2} = \{ [s, 0, i] \rightarrow [s, 0, t, 0, i] \mid t = \Theta(0, i) \} \]
\[ \cup \{ [s, 1, j] \rightarrow [s, 0, t, 1, j] \mid t = \Theta(1, j) \} \ldots \]

\[ F_1 = \{ [s, 0, i] \} \cup \{ [s, 1, j] \} \cup \{ [s, 2, k] \} \]

\[ F_2 = \{ [s, 0, t, 0, i] \mid t = \Theta(0, i) \} \cup \{ [s, 0, t, 1, j] \mid t = \Theta(1, j) \} \ldots \]

Executor After Full Sparse Tiling

```
for s=1,T
    for t=0,nt
        for i in sloop0(t)
            ... = ...Z[i]
        endfor
        for j in sloop1(t)
            Z[l'][j] = ...
            Z[r'][j] = ...
        endfor
        for k in sloop2(t)
            ... += Z[k]
        endfor
    endfor
endfor
```
Summary

- The original computation is specified with sets, scheduling functions, access functions, and dependences.
- Transformations are specified in terms of a transformation relations.
- Uninterpreted functions are associated with input domains, output ranges, run-time routines that will generate them, and symbolic relations that represent input to those routines.
- IEGen builds a MapIR to represent the executor and an Inspector Dependence Graph (IDG) to represent the inspector.
- After all transformations have been applied at compile time, the inspector and executor functions are generated.
Putting All the Pieces Together

- Subcomputations with indirect memory accesses will be identified with pragmas or an IDE.
- Sparse Polyhedral Framework (SPF) enables the specification of computations and run-time reordering transformations.
- Transformation writers will provide run-time libraries to compute sparse tilings, etc.
- The Inspector/Executor Generator (IEGen) will generate the inspector and executor that implement a specified sequence of transformations.
Parallelization using Inspector/Executor Strategies

Break computation that sweeps over mesh/sparse matrix into chunks/sparse tiles

Parallelism in Full Sparse Tiled Jacobi

Average parallelism = (# tiles) / (# tiles in critical path)

= 6 / 5 = 1.2
How can we increase parallelism between tiles?

- Order that tile growth is performed matters
- Best is to first grow tiles whose seed partitions are not adjacent

Improving Average Parallelism Using Coloring

= Create a partition graph
Improving Average Parallelism Using Coloring

- Create a partition graph
- Color the partition graph

Colorado State University

Renumber partitions consecutively by color
Grow Using New Partition Order

- Renumber the seed partition cells based on coloring
- Grow tiles using new ordering
- Notice that tiles 0 and 1 may be executed in parallel

Re-grow Using New Partition Order

- Renumber the seed partition cells based on coloring
- Grow tiles using new ordering
- Notice that tiles 0 and 1 may be executed in parallel
- Tiles 4 and 5 may also be executed in parallel
Average Parallelism is Improved

Average parallelism = (# tiles) / (# tiles in critical path)
= 6 / 3 = 2

Performance Evaluation

- Comparing OpenMP with blocking and dynamic scheduling versus full sparse tiling
  - OpenMP is simpler to express in original code, less overhead
  - FST has better temporal locality and asynchronous parallelism

- Comparing various programming models for specification and execution of arbitrary task graphs
  - TBB – Threading Building Blocks
  - Pthreads
  - OpenMP Tasks and OpenMP frontiers in task graph
  - CnC – Concurrent Collections
Full Sparse Tiling Helps When Have Lower Available Bandwidth

- **4 cores, pwtk matrix, 2000 iterations, 13.2 GB/sec Triad BW**

- **16 cores, pwtk matrix, 2000 iterations, 30.8 GB/s Triad BW**
Expressing Arbitrary Graphs in Various Programming Models

Comparison of Programming Models
8 cores, FST Jacobi, pwtk matrix

Conclusions

- Sparse Polyhedral Framework (SPF) provides abstractions needed to automate performance transformation of irregular/sparse apps
- Inspector/executor code generator (IEGen) will provide an approach for semi-automating the application of inspector/executor strategies
- Sparse tiling is an inspector/executor strategy that …
  - turns data reuse into data locality
  - results in arbitrary task graphs at runtime
  - enables putting off the point at which bandwidth bound computations quit scaling
Contributors

- Original SPF concept developed in collaboration with Larry Carter and Jeanne Ferrante
- Chris Krieger – Task graph programming model and possibly composition of prog models
- Geri George – Project planning and management
- Alum: Alan LaMielle – IEGen prototype
- Alum: Jon Roelofs – IEGenCC tool prototype