Review: Parallel Programming Models

• *Shared Memory Programming*
  - Start a single process and fork threads
  - Threads carry out work
  - Threads communicate through shared memory
  - Threads coordinate through synchronization

• *Distributed Memory Programming*
  - Start multiple processes on multiple systems
  - Processes carry out work
  - Processes communicate through message-passing
  - Processes coordinate either through message-passing or synchronization (generates messages)
Review: Shared Memory Programming

- **PThreads — Portable Operating System Interface [for Unix] (POSIX)**
  - Relatively low level
    - Programmer expresses thread management and coordination
    - Programmer decomposes parallelism and manages schedule
  - Most widely used for systems-oriented code, and also used for some applications

- **OpenMP**
  - Higher-level support for scientific programming on shared memory architectures
    - Programmers identifies parallelism and data properties, and guides scheduling at a high level
    - System decomposes parallelism and manages schedule
  - Arose from a variety of architecture-specific pragmas
Review: Shared Memory Programming

- **PThreads — Portable Operating System Interface [for Unix] (POSIX)**
  * Relatively low level
    - Programmer expresses thread management and coordination
    - Programmer decomposes parallelism and manages schedule
  * Most widely used for systems-oriented code, and also used for some applications

- **OpenMP**
  * Higher-level support for scientific programming on shared memory architectures
    - Programmers identifies parallelism and data properties, and guides scheduling at a high level
    - System decomposes parallelism and manages schedule
  * Arose from a variety of architecture-specific pragmas
OpenMP

- The virtually standard API for scientific shared memory parallel applications in C, C++ and Fortran

- Consists of
  - Compiler directives
  - Runtime routines
  - Environment variables

- Specifically maintained by OpenMP Architecture Review Board ([www.openmp.org](http://www.openmp.org))

- Latest version: 4.0
An Example

For-loop with independent iterations

```
for (i = 0; i < n; i++)
c[i] = a[i] + b[i];
```

For-loop parallelized using OpenMP pragma

```
#pragma omp parallel for 
  shared(n, a, b, c) 
  private(i)
for (i = 0; i < n; i++)
c[i] = a[i] + b[i];
```

% cc -xopenmp source.c
% setenv OMP_NUM_NUM_THREADS 4
% a.out
OpenMP Execution Model
Compiler does all the work here!
Guided parallelization!
Automatic Parallelization
Polyhedral Compilation Framework

A three-stage process

• Analysis: code $\rightarrow$ model, dependence analysis is the key

• Model transformation: sequential $\rightarrow$ parallel or other

• Code generation: model $\rightarrow$ code, parallel code or other
Polyhedral Compilation: Code to Model

Iteration domain:

- A set of \textit{n-dimensional} vectors
  - \textbf{Iteration Vector}: $\bar{x} = (i, j)$
  - Iteration domain: the set of values of $\bar{x}$

- **Polytopes** model sets of totally ordered n-dimensional vectors
- The set must be convex
- Example:

\begin{verbatim}
Code
for ( i=0; i < 5; i++ )
  for ( j=0; j < 5; j++ )
      ....
      endfor
endfor
\end{verbatim}

Iter. Domain Graph
**Definition: Convex Set**

Let $\mathbb{R}$ be a vector space over the real numbers. A set $S$ in $\mathbb{R}$ is **convex** iff, $\forall \mu, \lambda \in S$ and given any $t \in [0,1]$, the point

$$(1 - t) \cdot \mu + t \cdot \lambda \in S$$

**In words:**

Drawing a line segment between any two points of $S$, each point of this segment is also in $S$. 
Loops that have convex iteration domain

A statement surrounded by loops with unit-stride, no conditional and constant loop bounds has a convex iteration domain, if the iteration domain exists.

```
for ( i=0; i < N; i++ )
...
```

In general, in loops:

- The polytope set must be convex
- Convexity is the **central concept** of polyhedral optimization
Affine Function

Definition
A function $f: \mathbb{K}^m \to \mathbb{K}^n$ is affine if there exists a vector $\vec{b} \in \mathbb{K}^n$ and matrix $A \in \mathbb{K}^{n \times m}$ such that:

$$\forall \vec{x} \in \mathbb{K}^m, f(\vec{x}) = A\vec{x} + \vec{b}$$

Affine half-space definition
An affine half-space of $\mathbb{K}^m$ is defined as the set of points:

$$\{ \vec{x} \in \mathbb{K}^m | \vec{a} \vec{x} \leq \vec{b} \}$$
Example of an affine half-space:

\[ 3x_1 + 2x_2 \geq 11 \]

\[ 3x_1 + 2x_2 \leq 11 \]

\[ 3x_1 + 2x_2 = 11 \]
A special class of loop:
The lowerbound and upper bound of the loop are expressed as affine expressions of the surrounding loop variables and symbolic constants.

Given
\[
\text{do } i_1 = L_1, U_1 \\
\ldots \\
\text{do } i_n = L_n, U_n \\
S_1 : \ldots \\
S_2 : \ldots
\]

where \( L_m = f_m(i_1, i_2, \ldots, i_{m-1}) \) and \( f_m(\ldots) \) are all affine functions of variables \((i_1, \ldots, i_{m-1})\).
Modeling Affine Loop Iteration Domains

Corresponding loop constructs: unit stride, no conditionals

- Polytope dimension: number of surrounding loops
- Constraints: set by loop bounds
- Let’s all formulate it in this way: $A \cdot x - b \geq 0$
- Add symbolic constants if necessary

**Previous Loop Example**

```plaintext
for ( i = 0; i ≤ 5; ++i )
    for ( j = 0; j ≤ 5; ++j )
        A[i][j] = i * j;
```

$$D_R : \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} 0 \\ 5 \\ 0 \\ 5 \end{pmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -1 & 0 & 5 \\ 0 & 1 & 0 \\ 0 & -1 & 5 \end{bmatrix} \cdot \begin{pmatrix} i \\ j \\ 1 \end{pmatrix} \geq \bar{0}$$
Lexical Graphical Order

Given \( I = (i_1, i_2, \ldots, i_n) \) and \( I' = (i'_1, i'_2, \ldots, i'_n) \),
\( I < I' \) iff
\[ (i_1, i_2, \ldots, i_k) = (i'_1, i'_2, \ldots, i'_k) \) & \( i_{k+1} < i'_{k+1} \)

An Example of Loop

\[
\text{for ( } i = 0; i < 3; ++i \text{ )}
\]
\[
\text{for ( } j = 0; j < 3; ++j \text{ )}
\]
\[
A[i][j] = i \times j;
\]

Loop execution:

\[
(0,0)
\]
\[
(0,1)
\]
\[
(0,2)
\]
\[
(1,0)
\]
\[
(1,1)
\]
\[
(1,2)
\]
\[
\ldots
\]
**Definition: Bernstein conditions**

Given two references, there exists a dependence between them if the three following conditions hold:

- They reference the same array (cell)
- One of them is a write
- The two associated statements are executed

**Review: different types of dependences:**

- RAW (Read-After-Write): parallelization analysis
- WAR (Write-After-Read): parallelization analysis
- WAW (Write-After-Write): parallelization analysis
- RAR (Read-After-Read): data locality/reuse computations
Definition:
A statement $S$ depends on a statement $R$ (written $R \rightarrow S$) if there exists an operation $S(x^R_S)$, $R(x^R_R)$, a memory location $m$, and $x^R_S$, $x^R_R$ are respectively two iteration instances such that:

- $S(x^R_S)$ and $R(x^R_R)$ refer to the same memory location $m$, and at least one of them writes to that location
- $x^R_S$ and $x^R_R$ belongs to the iteration domain of $R$ and $S$,
- In the original sequential order, $R(x^R_R)$ is executed before $S(x^R_S)$. 
Linear Constraints Derived by Dependence Relationship

- $S(\vec{x}_S)$ and $R(\vec{x}_R)$ reference the same memory location:
  $$F_s \vec{x}_S + a_S = F_R \vec{x}_R + a_R.$$

- $\vec{x}_R$ and $\vec{x}_S$ within loop iteration domains:
  $$A_S \vec{x}_S + c_S \geq 0 \text{ and } A_R \vec{x}_R + c_R \geq 0.$$

- Precedence order: $R(\vec{x}_R)$ happens before $S(\vec{x}_S)$ for a particular dependence loop level $L$ where the dependence happens,

  for $i < L$: $x_{R,i} = x_{S,i}$

  for $i = L$:

  $$x_{R,L} \leq x_{S,L}$$

These linear equalities and inequalities can be rewritten as:

$$P_{I,S}\vec{x}_S - P_{I,R}\vec{x}_R + b \geq 0.$$
The dependence polyhedron for $\mathcal{R} \rightarrow \mathcal{S}$ at a given level $l$ and for a given pair of references $f_R, f_S$ is described as [Feautrier/Bastoul]:

$$\mathcal{D}_{R,S,f_R,f_S,l} :$$

$$D \left( \begin{array}{c} \vec{x}_S \\ \vec{x}_R \end{array} \right) + d = \begin{bmatrix} F_S & -F_R \\ A_S & 0 \\ 0 & A_R \\ P_S & -P_R \end{bmatrix} \left( \begin{array}{c} \vec{x}_S \\ \vec{x}_R \end{array} \right) + \begin{pmatrix} \frac{a_S - a_R}{c_S} \\ c_R \\ b \end{pmatrix} = 0 \geq 0$$
Dependence Polyhedron Example

The example loop

```c
for ( i = 1; i <= 3; ++i ) {
    S1: a[i] = i;
    for ( j = 1; j <= 3; ++j )
        S2: b[j] = (b[j] + a[i])/2;
}
```

\[ D_{S_2 \delta S_2, \langle b[j_{S_2'}], b[j_{S_2}] \rangle, 1} : \]

\[
\begin{bmatrix}
0 & 1 & 0 & -1 & 0 \\
1 & 0 & 0 & 0 & -1 \\
-1 & 0 & 0 & 0 & 3 \\
0 & 1 & 0 & 0 & -1 \\
0 & -1 & 0 & 0 & 3 \\
0 & 0 & 1 & 0 & -1 \\
0 & 0 & -1 & 0 & 3 \\
0 & 0 & 0 & 1 & -1 \\
0 & 0 & 0 & -1 & 3 \\
-1 & 0 & 1 & 0 & -1 \\
\end{bmatrix} \left( \begin{array}{c}
 i_{S_2'} \\
 j_{S_2'} \\
 i_{S_2} \\
 j_{S_2} \\
 1 \\
\end{array} \right) \left( \begin{array}{c}
= \vec{0} \\
\geq \vec{0} \\
\end{array} \right)
\]
Algorithm for Finding Dependence Polyhedron

Step by step:

Initialize DDG graph with one node for every statement
For each pair of statements $R, S$
For each pair of references $f_R, f_S$
For each loop level $l$: min_depth to common_depth
Build dependence polyhedron $D_{R,S,f_R,f_S,l}$
If it is not empty, then get the dependence type
Add edge($R, S, \{l, D_{R,S,f_R,f_S,l,\text{type}}\}$)
The example loop

```plaintext
for ( i = 1; i <= 3; ++i ) {
  S1: a[i] = i;
    for ( j = 1; j <= 3; ++j )
      S2: b[j] = (b[j] + a[i])/2;
}
```

![Data Dependence Graph Example](image)
One Dimensional Affine Transformation

Assign a time stamp to every statement instance:

Definition: Affine Schedule
Given a statement $S$, a $p$-dimensional affine schedule $\theta_S$ is an affine form on the outer loop iterators $\vec{x}_S$ and the global parameters $\vec{n}$. It is written:

$$\theta_S(\vec{x}_S) = T_S \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}$$
Definition: Precedence condition
Given $\theta_R$ a schedule for the instances of $R$, $\theta_S$ a schedule for the instances of $S$. $\theta_R$ and $\theta_S$ are legal schedules if $\forall (\vec{x}_R, \vec{x}_S) \in D_{R,S}$

$$\theta_R(\vec{x}_R) \prec \theta_S(\vec{x}_S)$$

- Objective function:
  Find the best schedule such that it can:
  - Minimize latency
  - Maximize fine grained parallelism
  - Enhance locality
  - ...

An Affine Schedule Example

for $i = 0, n$
  S1: $s(i) = 0$
  for $j = 0, n$
    S2: $s(i) = s(i) + a(i,j) \times x(j)$
  endfor
endfor

- Objective function
  Minimize $\theta_{last}$
- Possible affine schedules
  One is:
  $\theta(1, i) = 0$
  $\theta(2, i, j) = j + 1$
  Another is:
  $\theta(1, i) = i$
  $\theta(2, i, j) = i + j + 1$
Find An Affine Schedule

We start with one dimensional time schedule:

- Find legal affine schedules
- Find best schedule according to the objective function
A naive Approach

- For all pairs of \((\vec{x}_R, \vec{x}_S) \in D_{R,S}\)
- Replace the variables \(\vec{x}\) with the values of dependence instances above to get a linear system of inequalities
  \[
  \theta_R(\vec{x}_R) + 1 \leq \theta_S(\vec{x}_S)
  \]
- Solve for the affine coefficients of \(\theta_R\) and \(\theta_S\)
  \[
  \theta_R(\vec{x}) = \vec{a}_R \vec{x} + b_R
  \]
- Challenge: too many pairs of statement instances in dependence
Theorem
Let $\mathcal{D}$ be a non-empty polyhedron defined by $p$ affine inequalities

$$a_k x^l + b_k \geq 0, \ k = 1, p$$

Then an affine form $\phi$ is nonnegative everywhere in $\mathcal{D}$ iff it is a positive affine combination:

$$\phi(x) = \lambda_0 + \sum_k \lambda_k (a_k x^l + b_k), \ \lambda_k \geq 0$$

The set of $\lambda_k$ coefficients are called Farkas multipliers.
Application of Farkas Lemma

Two conditions:

- $\theta(\vec{x})$ is non-negative
- Precedence condition, for dependence polyhedron:
  $$\theta_S(\vec{x}_S) - \theta_R(\vec{x}_R) \geq 0$$

Two steps:

- Equate the coefficients of $x$ variables and $x$ variables are gone
- Use Fourier-Motzkin elimination to eliminate as many Farkas multipliers as possible so that only affine schedule coefficients are left

Now we only have the affine scheduling coefficients as unknown variables. We add around linear times of $p$ inequalities, with $p$ being the dimension of dependence polyhedron compared to $2^p$. 
Using Farkas Lemma

Assume iteration polyhedron:

\[ a_{Sk} \begin{pmatrix} x \\ n \end{pmatrix} + b_{Sk} \geq 0, \ k = 1, m_s \]

Dependence polyhedron:

\[ c_{ek} \begin{pmatrix} x \\ y \\ n \end{pmatrix} + d_{ek} \geq 0, \ k = 1, m_e \]
Using Farkas Lemma

Non-negative schedule constraint:

\[ \theta(S, x) = \mu_{s0} + \sum_{k=1}^{ms} \mu_{sk}(a_{sk} \begin{pmatrix} x \\ n \end{pmatrix} + b_{sk}) \]

Non-negative precedence constraint:

\[ \theta(S, y) - \theta(R, x) - 1 = \lambda_{e0} + \sum_{k=1}^{me} \lambda_{ek} \left( c_{ek} \begin{pmatrix} x \\ y \\ n \end{pmatrix} + d_{ek} \right) \]
Using Farkas Lemma — Example

1. for $i = 0, n$
   
   $S1: s(i) = 0$
   
   for $j = 0, n$
   
   $S2: s(i) = s(i) + a(i,j) \times (j)$
   
   endfor
   
   endfor

   Schedules may be written as:
   
   $\theta(1, i) = \mu_{2,0} + \mu_{1,1}i + \mu_{1,2}(n-i)$
   
   $\theta(2, i, j) = \mu_{2,0} + \mu_{2,1}i + \mu_{2,2}(n-i) + \mu_{2,3}j + \mu_{2,4}(n-j)$

   Two dependences in DDG:
   
   $S1 \prec S2$ when $j = 0$
   
   $S2(j') \prec S2(j)$ when $j' = j - 1$

2. For the first dependence edge in DDG, use Farkas Lemma:
   
   $\mu_{2,0} + \mu_{2,1}i + \mu_{2,2}(n-i) + \mu_{2,3}j + \mu_{2,4}(n-j)$
   
   $-(\mu_{1,0} + \mu_{1,1}i + \mu_{1,2}(n-i)) - 1$
   
   $= \lambda_{1,0} + \lambda_{1,1}i + \lambda_{1,2}(n-i) + \lambda_{1,3}j + \lambda_{1,4}(n-j) - \lambda_{1,5}j$

   For the second dependence edge in DDG, it is uniform dependence, which is:
   
   $\mu_{2,0} + \mu_{2,1}i + \mu_{2,2}(n-i) + \mu_{2,3}j + \mu_{2,4}(n-j)$
   
   $-(\mu_{2,0} + \mu_{2,1}i + \mu_{2,2}(n-i) + \mu_{2,3}(j-1) + \mu_{2,4}(n-j+1)) - 1$

   And we have,
   
   $\mu_{2,3} - \mu_{2,4} - 1 \geq 0$

   We do not apply Farkas Lemma since it does not depend on loop iterators and therefore does not depend on any constraints on the loop iteration domain.

3. By a process of identification for loop iterator coefficients, we have:
   
   $\mu_{2,0} - \mu_{1,0} - 1 = \lambda_{1,0}$
   
   $\mu_{2,1} - \mu_{2,2} - \mu_{1,1} + \mu_{1,2} = \lambda_{1,1} - \lambda_{1,2}$
   
   $\mu_{2,3} - \mu_{2,4} = \lambda_{1,3} - \lambda_{1,4} - \lambda_{1,5}$
   
   $\mu_{2,2} + \mu_{2,4} - \mu_{1,2} = \lambda_{1,2} + \lambda_{1,4}$

   Eliminate as much as unknown as possible. One possible result:
   
   $\lambda_{1,0} = \mu_{2,0} - \mu_{1,0} - 1 \geq 0$
   
   $\lambda_{1,1} = \mu_{2,1} + \mu_{2,4} - \mu_{1,1} - \lambda_{1,4} \geq 0$
   
   $\lambda_{1,3} = \mu_{2,3} - \mu_{2,4} + \lambda_{1,4} + \lambda_{1,5} \geq 0$
   
   $\lambda_{1,2} = \mu_{2,2} + \mu_{2,4} - \mu_{1,2} - \lambda_{1,4} \geq 0$
   
   $\mu_{2,3} - \mu_{2,4} - 1 \geq 1$

   Finally we have:

   $0 \leq \mu_{1,1} \leq \mu_{2,1} + \mu_{2,4}$
   
   $0 \leq \mu_{1,2} \leq \mu_{2,2} + \mu_{2,4}$
   
   $\mu_{2,0} \geq 1 + \mu_{1,0}$
   
   $\mu_{2,3} \geq 1 + \mu_{2,4}$
Legitimate Schedules

Constraints on $\mu$ coefficients:

\[ 0 \leq \mu_{1,1} \leq \mu_{2,1} + \mu_{2,4} \]
\[ 0 \leq \mu_{1,2} \leq \mu_{2,2} + \mu_{2,4} \]
\[ \mu_{2,0} \geq 1 + \mu_{1,0} \]
\[ \mu_{2,3} \geq 1 + \mu_{2,4} \]

Schedules that satisfy precedence (causality) condition:
\[ \theta(1, i) = \mu_{1,0} + \mu_{1,1}i + \mu_{1,2}(n - i) \]
\[ \theta(2, i, j) = \mu_{2,0} + \mu_{2,1}i + \mu_{2,2}(n - i) + \mu_{2,3}j + \mu_{2,4}(n - j) \]

Possible schedules: practice - how the $\mu$ coefficients are set?
\[ \theta(1, i) = 0, \ \theta(2, i, j) = j + 1 \]
Or:
\[ \theta(1, i) = i, \ \theta(2, i, j) = i + j + 1 \]
Finding An Optimal Schedule

Lemma
If all domains are bounded, and if there exists at least one affine schedule \( \theta \), then there exists at least one affine form in the structure parameters:

\[ L = h \cdot n + k \]

such that:

\[ \forall S, x \in D_S : L - \theta(S, x) \geq 0 \]

By Farkas Lemma, we have:

\[ L - \theta(S, x) = v_{S0} + \sum_k v_{Sk} \left( a_{sk} \left( \frac{x}{n} \right) + b_{sk} \right) \]
We add new unknowns, can solve for the minimal of $L$ by considering vector $h$ as leading unknown variables. We can try to minimize $h$ as much as possible, therefore we can use lexicographical as suboptimal solutions. PIP gives lexicographical minimal, not necessarily optimal, depending on the ordering of structure variables.

Why lexicographical solution not necessarily optimal?
Multi-dimensional Affine Schedule

Recall definition:
Given a statement $S$, a $p$-dimensional affine schedule $\theta_S$ is an affine form on the outer loop iterators $\vec{x}_S$ and the global parameters $\vec{n}$. It is written:

$$\theta_S(\vec{x}_S) = T_S \begin{pmatrix} \vec{x}_S \\ \vec{n} \\ 1 \end{pmatrix}$$

For precedence rule, if one dimensional, we can simply do:

$$\theta_S(\vec{x}_S) \geq \theta_R(\vec{x}_R) + 1 \text{ if } \theta_R(\vec{x}_R) < \theta_S(\vec{x}_S)$$

Does this inequality apply for multi-dimensional affine schedule at every component of the vector?
A Naive Solution

- Starting from the first component in vector $\theta_S$ and $\theta_R$
- Solve for $\theta_S(i) \geq \theta_R(i)$ as if it is one dimensional schedule
- If there exists solution $\theta_S(i) = \theta_R(i)$, then go to the next component by incrementing $i$, go back to last step
- If there does not exist solution $\theta_S(i) = \theta_R(i)$, terminate here

This approach is complete, however the search space is explosive.
An Improved Solution

- Step 1: Solve $\theta_S(i) > \theta_R(i)$ for some of the dependence edges in DDG
- Step 2: Then solve for the rest dependence edges in DDG by refining solutions from the first step
- Step 3: Repeat Step 1 and Step 2 until all dependence edges are satisfied

There is a termination condition proof from "Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time", [Feautrier IJPP'92]

Other possible greedy solutions for different objectives? Open problem ...
References

- ”Code Generation in the Polyhedral Model Is Easier Than You Think”, PACT’04, Cédric Bastoul
- ”Parametrized Polyhedra and Enumerating their Vertices”, Widle et al
- ”Parametric integer programming”, P. Feautrier
- ”Counting Integer Points in Parametric Polytopes using Barvinoks Rational Functions”, Verduolaeg et al.
- ”Some Efficient Solutions to the Affine Scheduling Problem. Part I. One dimensional Time”, Feautrier et al.
- ”Some Efficient Solutions to the Affine Scheduling Problem. Part II. Multidimensional Time”, Feautrier et al.
Tools

- PoCC: the Polyhedral Compiler Collection
  http://www.cse.ohio-state.edu/~pouchet/software/pocc/

- Tools within this package:
  Candl (dependence analysis)
  Clan (IR extraction)
  PIPLib (Parametric Integer Programming)
  Pluto (Scheduling and Tiling)

- Example: matmul program
  At ilab machines:
  /ilab/users/zz124/cs671_2013/poly/candl_examples