Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model

Uday Bondhugula

Department of Computer Science & Engineering
The Ohio State University

U. Bondhugula, M. Baskaran, S. Krishnamoorthy, A. Rountev, P. Sadayappan (OSU), J. Ramanujam (LSU)

April 3, 2008

Multicore architectures

- Architectures with multiple processing units on chip have become mainstream
  - General-purpose multicore microprocessors
  - Specialized: GPUs, Cell, MPSoCs
- Difficulty of Parallel Programming
- **Automatic Parallelization**: user does nothing

Multicore architectures

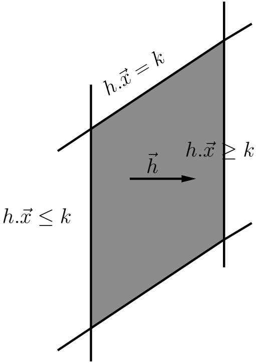
- Architectures with multiple processing units on chip have become mainstream
  - General-purpose multicore microprocessors
  - Specialized: GPUs, Cell, MPSoCs
- Difficulty of Parallel Programming
- **Automatic Parallelization**: user does nothing

Multicore architectures

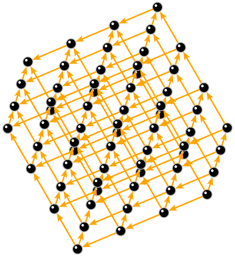
- Architectures with multiple processing units on chip have become mainstream
  - General-purpose multicore microprocessors
  - Specialized: GPUs, Cell, MPSoCs

- Difficulty of Parallel Programming
- **Automatic Parallelization**: user does nothing

Background: polyhedral/polytope model



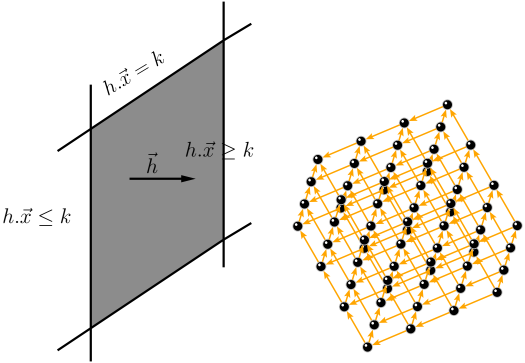
(a) A hyperplane



(b) A polytope

- Loop nests with regular accesses (statically predictable) - sequences of imperfectly nested loops
- More general code like non-affine accesses, dynamic control can also be handled with conservative assumptions

Background: polyhedral/polytope model



(c) A hyperplane (d) A polytope

- Loop nests with regular accesses (statically predictable) - sequences of imperfectly nested loops
- More general code like non-affine accesses, dynamic control can also be handled with conservative assumptions

Polyhedral compiler framework

1. Dependence analysis (exact affine dependences) [Feautrier91, Pugh92, Vasilache06ICS]
2. Automatic transformations [Feautrier92, Lim/Lam97, Griebl04]
3. Code generation from specified transforms [Omega90s, Quilleré00, **Bastoul04, CLooG**, Vasilache06CC]

- Significant advances in first and last step during this decade
- Semi-automatic approaches demonstrating polyhedral model as a powerful representation [Cohen05ICS, Girbal06IJPP]

Polyhedral compiler framework

1. Dependence analysis (exact affine dependences) [Feautrier91, Pugh92, Vasilache06ICS]
2. Automatic transformations [Feautrier92, Lim/Lam97, Griebl04]
3. Code generation from specified transforms [Omega90s, Quilleré00, **Bastoul04, CLooG**, Vasilache06CC]

- Significant advances in first and last step during this decade
- Semi-automatic approaches demonstrating polyhedral model as a powerful representation [Cohen05ICS, Girbal06IJPP]

Polyhedral optimization

1. Dependence analysis (exact affine dependences) [Feautrier91, Pugh92, Vasilache06ICS]
2. **Automatic transformations (for parallelism and locality)**
3. Code generation from specified transforms [Omega90s, Quilleré00, Bastoul04, CLooG, Vasilache06CC]

Our work: a new theoretical framework for automatic transformation

Polyhedral model: an example

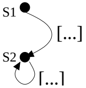
```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    S1: A[i,j] = A[i,j]+u1[i]*v1[j] + u2[i]*v2[j];

for (i''=0; i'<N; i'++)
  for (j'=0; j'<N; j'++)
    S2: x[i'] = x[i']+A[j',i']*y[j'];
```

original code

$$D^{S_1}\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 1 & -1 \\ 0 & -1 & 1 & -1 \end{pmatrix}\begin{pmatrix} i \\ j \\ N \\ 1 \end{pmatrix} \geq 0$$

(1.1) Statement domain



(2) The Generalized Dependence Graph

$$\left(\begin{array}{cccccc} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & -1 \\ \hline 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{array}\right)\begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{array}{c} \geq 0 \\ \geq 0 \\ \geq 0 \\ \geq 0 \\ \hline = 0 \\ = 0 \end{array}$$

(3.1) An exact dependence polyhedron (S1→S2)

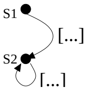
Polyhedral model: Motivation for automatic transformation

GEMVER

```
dcopy(m * n, A, B, 1);
dger(m, n, 1.0, u1, 1, v1 , 1, B, m);
dger(m, n, 1.0, u2, 1, v2 , 1, B, m);
dcopy(n, z, x, 1);
dgemv('T', m, n, beta, B, m, y, 1, 1.0, x, 1);
dgemv('N', m, n, alpha, B, m, x, 1, 0.0, w, 1);
```

BLAS version [Siek et al, POHLL '08]

$$B = A + u_1 v_1^T + u_2 v_2^T$$
$$x = \beta B^T y + z$$
$$w = \alpha B x$$



The Generalized Dependence Graph

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & -1 \\ \hline 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{pmatrix} \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{matrix} \geq 0 \\ \geq 0 \\ \geq 0 \\ \geq 0 \\ = 0 \\ = 0 \end{matrix}$$

An exact dependence polyhedron (S1→S2)

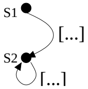
Polyhedral model: Motivation for automatic transformation

GEMVER

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    S1: A[i,j] = A[i,j]+u1[i]*v1[j] + u2[i]*v2[j];

for (i'=0; i'<N; i'++)
  for (j'=0; j'<N; j'++)
    S2: x[i'] = x[i']+A[j',i']*y[j'];
```

original code



The Generalized Dependence Graph

	S1			S2			
	i	j	$const$	i	j	$const$	
c_1	0	1	0	1	0	0	parallel
c_2	1	0	0	0	1	0	fwd_dep
c_3	0	0	0	0	0	1	scalar

Statement-wise transformation

$$
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 1 & -1 \\
0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 1 & -1 \\
\hline
1 & 0 & 0 & -1 & 0 & 0 \\
0 & 1 & -1 & 0 & 0 & 0
\end{pmatrix}
\begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix}
\begin{matrix} \geq 0 \\ \geq 0 \\ \geq 0 \\ \geq 0 \\ \hline = 0 \\ = 0 \end{matrix}
$$

An exact dependence polyhedron (S1→S2)

```
for (c1=0; c1<N; c1++)
  for (c2=0; c2<N; c2++)
    A[c2,c1] = A[c2,c1]+u[c2]*v[c1];
    x[c1] = x[c1]+A[c2,c1]*y[c2];
```

Transformed code (not final)

Cores	Our (poly)	native cc	ACML 4.0.1/ifort
1	0.348s	2.33s	0.679s
2	0.238s	1.46s	0.59s

AMD Opteron (dual core) 2.6 GHz, execution time

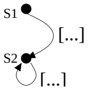
Polyhedral model: Motivation for automatic transformation

GEMVER

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    S1: A[i,j] = A[i,j]+u1[i]*v1[j] + u2[i]*v2[j];

for (i'=0; i'<N; i'++)
  for (j'=0; j'<N; j'++)
    S2: x[i'] = x[i']+A[j',i']*y[j'];
```

original code



The Generalized Dependence Graph

	S1 i	S1 j	S1 *const*	S2 i	S2 j	S2 *const*	
c_1	0	1	0	1	0	0	parallel
c_2	1	0	0	0	1	0	fwd_dep
c_3	0	0	0	0	0	1	scalar

Statement-wise transformation

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & -1 \\ \hline 1 & 0 & 0 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \end{pmatrix} \begin{bmatrix} i \\ j \\ i' \\ j' \\ N \\ 1 \end{bmatrix} \begin{matrix} \geq 0 \\ \geq 0 \\ \geq 0 \\ \geq 0 \\ \hline = 0 \\ = 0 \end{matrix}$$

An exact dependence polyhedron (S1→S2)

```
for (c1=0; c1<N; c1++)
  for (c2=0; c2<N; c2++)
    A[c2,c1] = A[c2,c1]+u[c2]*v[c1];
    x[c1] = x[c1]+A[c2,c1]*y[c2];
```

Transformed code (not final)

Cores	Over native cc	Over vendor BLAS
1	6.7x	2.0x
2	6.1x	2.5x

AMD Opteron (dual core) 2.6 GHz, Poly Speedup

Affine Transformations in the polyhedral model

A one-dimensional affine transform for statement S_k is defined by:

$$\begin{aligned}\phi_{S_k}(\vec{i}) &= \begin{bmatrix} c_1 & c_2 & \dots & c_{m_{S_k}} \end{bmatrix} \left(\vec{i} \right) + c_0 \\ &= \begin{bmatrix} c_1 & c_2 & \dots & c_{m_{S_k}} & c_0 \end{bmatrix} \begin{pmatrix} \vec{i} \\ 1 \end{pmatrix}\end{aligned}$$

where $[c_0, c_1, c_2, \dots, c_{m_{S_k}}] \in \mathcal{Z}$.

- An affine transform $\equiv$ A new scanning hyperplane $\equiv$ A loop in the transformed space (with a particular property)

Affine transformations

- A transformation for each statement, S: $T_S\vec{i} + \vec{b_S}$

$$\begin{pmatrix} i'_1 \\ i'_2 \\ i'_3 \\ \vdots \\ i'_n \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{pmatrix} \begin{pmatrix} i_1 \\ i_2 \\ i_3 \\ \vdots \\ i_n \end{pmatrix} + \begin{pmatrix} c_{01} \\ c_{02} \\ c_{03} \\ \vdots \\ c_{0n} \end{pmatrix}$$

- Full column-ranked transform is a one-to-one mapping
- Each 1-d transform (ϕ) can later be marked as a space loop or time (sequential) loop or a band of them can be tiled
- **Problem**: how do you find good transformations optimized for parallelism and locality?

Tiling in the polyhedral model

- Tile: a portion of the iteration space that can be executed atomically
- **Tiling for parallelism:** Enables coarse-grained parallelization: reduces frequency of communication
- **Tiling for locality:** Allows reuse along multiple dimensions - tile fits in faster memory
- Tile shape and size affect the volume and frequency of communication / number of cache misses
- Legality of tiling for restricted input and/or weaker dependence abstractions are well understood [Irigoin and Triolet 88, Wolf/Lam 91, Darte et al. 97]

Tiling in the polyhedral model

- Tile: a portion of the iteration space that can be executed atomically
- **Tiling for parallelism:** Enables coarse-grained parallelization: reduces frequency of communication
- **Tiling for locality:** Allows reuse along multiple dimensions - tile fits in faster memory
- Tile shape and size affect the volume and frequency of communication / number of cache misses
- Legality of tiling for restricted input and/or weaker dependence abstractions are well understood [Irigoin and Triolet 88, Wolf/Lam 91, Darte et al. 97]

Tiling in the polyhedral model

- Tile: a portion of the iteration space that can be executed atomically
- **Tiling for parallelism:** Enables coarse-grained parallelization: reduces frequency of communication
- **Tiling for locality:** Allows reuse along multiple dimensions - tile fits in faster memory
- Tile shape and size affect the volume and frequency of communication / number of cache misses
- Legality of tiling for restricted input and/or weaker dependence abstractions are well understood [Irigoin and Triolet 88, Wolf/Lam 91, Darte et al. 97]

Tiling in the polyhedral model

- Tile: a portion of the iteration space that can be executed atomically
- **Tiling for parallelism:** Enables coarse-grained parallelization: reduces frequency of communication
- **Tiling for locality:** Allows reuse along multiple dimensions - tile fits in faster memory
- Tile shape and size affect the volume and frequency of communication / number of cache misses
- Legality of tiling for restricted input and/or weaker dependence abstractions are well understood [Irigoin and Triolet 88, Wolf/Lam 91, Darte et al. 97]

Tiling and its legality for exact polyhedral dependences

If $\vec{s}$ and $\vec{t}$ are dependent through dependence polyhedron P_e (corresponding to a dependence edge $s_i \rightarrow s_j$), then

$$\phi_{s_i}(\vec{t}) - \phi_{s_j}(\vec{s}) \quad \geq \quad 0, \quad \langle \vec{s}, \vec{t} \rangle \in P_e$$

- Extension of classic condition from Irigoin and Triolet [PoPL88]: dependence only has non-negative components along ϕ
- At least two independent ϕ's that satisfy the above property for all unsatisfied dependences so far $\rightarrow$ Tiling
- For affine dependences and statements of different dimensionalities (coming from arbitrarily nested loops)

Tiling and its legality for exact polyhedral dependences

If $\vec{s}$ and $\vec{t}$ are dependent through dependence polyhedron P_e (corresponding to a dependence edge $s_i \longrightarrow s_j$), then

$$\phi_{s_i}(\vec{t}) - \phi_{s_j}(\vec{s}) \quad \geq \quad 0, \quad \langle \vec{s}, \vec{t} \rangle \in P_e$$

- Extension of classic condition from Irigoin and Triolet [PoPL88]: dependence only has non-negative components along ϕ
- At least two independent ϕ's that satisfy the above property for all unsatisfied dependences so far $\longrightarrow$ Tiling
- For affine dependences and statements of different dimensionalities (coming from arbitrarily nested loops)

Capturing communication volume and reuse distance

- $\phi_{S_i}(\vec{t}) - \phi_{S_j}(\vec{s})$ is a very important affine function
- Define an affine form δ_e in $\vec{s}$, $\vec{t}$ (for every dependence):

$$\delta_e(\vec{s}, \vec{t}) = \phi_{s_i}(\vec{t}) - \phi_{s_j}(\vec{s}), \quad \langle \vec{s}, \vec{t} \rangle \in P_e$$

- Dot product of hyperplane with dependence ($\mathbf{h}.\vec{d}$)
- Number of hyperplane instances separating source and sink

Capturing communication volume and reuse distance

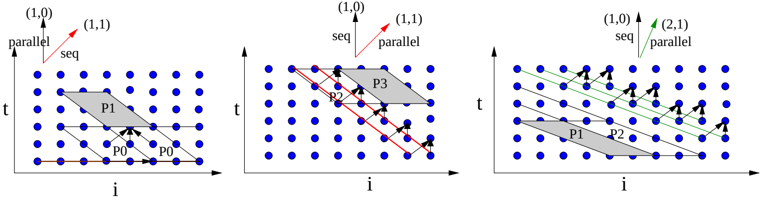
- $\phi_{S_i}(\vec{t}) - \phi_{S_j}(\vec{s})$ is a very important affine function
- Define an affine form δ_e in $\vec{s}$, $\vec{t}$ (for every dependence):

$$\delta_e(\vec{s}, \vec{t}) \quad = \quad \phi_{s_i}(\vec{t}) - \phi_{s_j}(\vec{s}), \quad \langle \vec{s}, \vec{t} \rangle \in P_e$$

- Dot product of hyperplane with dependence $(\mathbf{h}.\vec{d})$
- Number of hyperplane instances separating source and sink

Capturing communication volume and reuse distance

- Consider the stencil code below with deps: (1,0), (1,1), (1,-1)



- Represents the component of a dependence along the hyperplane (ϕ)
  - Communication volume (per unit area) at processor tile boundaries
  - Cache misses at local tile edges (L2, L1, registers)

Cost function in the polyhedral framework

Minimizing $\delta_e(\vec{s}, \vec{t})$ can be used to:

- Find hyperplane that minimizes inter-tile communication volume (rate per unit area)
- Find direction that minimizes reuse distance

But directly attempting to optimize δ_e is problematic

- Not expressible as a linear function of transformation coefficients
- $\phi(\vec{t}) - \phi(\vec{s})$ could be $c_1 i + (c_2 - c_3)j$, where $1 \leq i \leq N \wedge 1 \leq j \leq N \wedge i \leq j$

Using a bounding function

- Even if δ_e involves loop variables, it will still be less than a linear function of parameters

$$\begin{aligned} \phi_{s_i}(\vec{t}) - \phi_{s_j}(\vec{s}) &\leq \mathbf{u}.\vec{n} + w, \quad \langle \vec{s}, \vec{t} \rangle \in P_e \\ v(\vec{n}) \; - \; \delta_e(\vec{s}, \vec{t}) &\geq 0, \quad \langle \vec{s}, \vec{t} \rangle \in P_e, \quad \forall e \in E \end{aligned}$$

where $\vec{n}$ is the vector of program parameters

- Bound from above and minimize the coefficients of the bound after linearizing with Affine form of the Farkas Lemma
- The δ_e for each dependence is bounded in this manner

Using a bounding function

- Even if δ_e involves loop variables, it will still be less than a linear function of parameters

$$
\begin{aligned}
\phi_{s_i}(\vec{t}) - \phi_{s_j}(\vec{s}) \quad &\leq \quad \mathbf{u}.\vec{n} + w, \quad \langle \vec{s}, \vec{t} \rangle \in P_e \\
v(\vec{n}) \; - \; \delta_e(\vec{s}, \vec{t}) \quad &\geq \quad 0, \quad \langle \vec{s}, \vec{t} \rangle \in P_e, \quad \forall e \in E
\end{aligned}
$$

where $\vec{n}$ is the vector of program parameters

- Bound from above and minimize the coefficients of the bound after linearizing with Affine form of the Farkas Lemma
- The δ_e for each dependence is bounded in this manner

Bounding function approach (Farkas Lemma)

Now, use the affine form of the Farkas lemma on the bounding constraint

$$\mathbf{u}.\vec{n} + w - \delta_e(\vec{s}, \vec{t}) \quad \equiv \quad \lambda_{e0} + \sum_{k=1}^{m_e} \lambda_{ek} * f_{ek}(\vec{s}, \vec{t}), \quad f_{ek} \in P_e$$

- Linearizes legality and communication volume/reuse distance bounding constraints
- Use PIP to find the lexicographic minimal solution that satisfies above system with $\vec{u}$ and w in the leading position

$$\text{minimize}_{\prec} \left\{ \mathbf{u}, w, \ldots, c_i's, \ldots \right\}$$

- Minimizes the maximum dependence component along hyperplane normal (across all dependences)

Implications of cost function minimization

- $\mathbf{u} = 0, w = 0$: ϕ is a communication-free parallel hyperplane
- $\mathbf{u} = 0, w = const$: constant minimum boundary line communication/cache misses
- $\mathbf{u} > 0$: non-constant (large) amount of communication/cache misses

- A solution for multiple statements at a level is a fused loop
- Several refinements possible

Implications of cost function minimization

- $\mathbf{u} = 0, w = 0$: ϕ is a communication-free parallel hyperplane
- $\mathbf{u} = 0, w = const$: constant minimum boundary line communication/cache misses
- $\mathbf{u} > 0$: non-constant (large) amount of communication/cache misses

- A solution for multiple statements at a level is a fused loop
- Several refinements possible

Implications of cost function minimization

- $\mathbf{u} = 0, w = 0$: ϕ is a communication-free parallel hyperplane
- $\mathbf{u} = 0, w = const$: constant minimum boundary line communication/cache misses
- $\mathbf{u} > 0$: non-constant (large) amount of communication/cache misses

- A solution for multiple statements at a level is a fused loop
- Several refinements possible

Finding independent solutions iteratively

- Once a solution (a hyperplane for all statements is found), the same formulation is augmented with additional linear independence constraints
- Find independent solutions one after the other till enough hyperplanes to scan all statements are found
- Dependences are not removed as we proceed from hyperplane to another unless they need to be
    - A hierarchy of permutable loop nest sets is found (with a cost function and for any polyhedral input)
    - Fully tilable, partially tilable (tiling at any arbitrary level) or inner tilable loops are identified

Finding independent solutions iteratively

- Once a solution (a hyperplane for all statements is found), the same formulation is augmented with additional linear independence constraints
- Find independent solutions one after the other till enough hyperplanes to scan all statements are found
- Dependences are not removed as we proceed from hyperplane to another unless they need to be
    - A hierarchy of permutable loop nest sets is found (with a cost function and for any polyhedral input)
    - Fully tilable, partially tilable (tiling at any arbitrary level) or inner tilable loops are identified

Summary of Algorithm

- Affine dependences are pushed as much inside as possible
- Outer loops are space with minimal communication
  - Synchronization-free parallel loops end up first if they exist
  - Next, space loops with minimal communication are found
- Inner loops are sequential time with maximum reuse
- Linearly independent sub-space construction, avoiding trivial solutions: reasonable choices made to avoid combinatorial explosion
- **Complexity**: Very fast in practice

Summary of Algorithm

- Affine dependences are pushed as much inside as possible
- Outer loops are space with minimal communication
  - Synchronization-free parallel loops end up first if they exist
  - Next, space loops with minimal communication are found
- Inner loops are sequential time with maximum reuse
- Linearly independent sub-space construction, avoiding trivial solutions: reasonable choices made to avoid combinatorial explosion
- **Complexity**: Very fast in practice

The PLuTo system

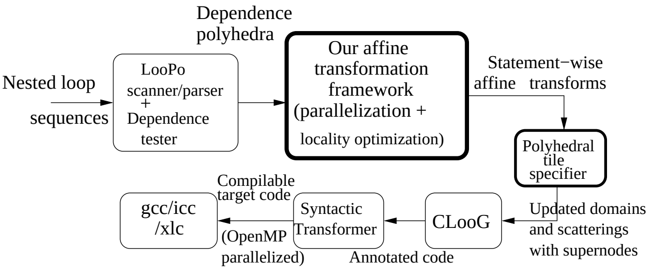


Figure: The PLuTo automatic parallelizer (source-to-source)

- Framework implemented with PipLib and interfaced with LooPo frontend (Univ. of Passau, Germany) and CLooG (Cédric Bastoul, INRIA Saclay)

Transformation framework running time

Code	Num of statements	Num of loops	Num of deps	Running time
2-d Jacobi	2	6	20	0.05s
Haar 1-d	3	5	12	0.018s
LU	2	5	10	0.022s
TCE 4-index	4	20	15	0.20s
Swim	58	160	639	20.9s

Table: Transformation tool running time

Experimental results: preview

- Intel Core2 Quad Q6600 2.4 GHz (quad core), DDR2 667 RAM
- 32 KB L1 cache, 8 MB (shared) L2 cache (4MB per core pair)
- ICC 10.1 (-fast), Linux 2.6.18 x86-64

Summary of performance improvement

Table: Improvement over state-of-the-art research compiler frameworks and native production compiler

Benchmark	Single core improvement		Multi-core speedup (4 cores)	
	over native compiler	over state-of-the-art research	over native compiler	over state-of-the-art research
Jacobi stencil (imperfect)	5.23x	2.1x	20x	2.7x
2-d FDTD	3.7x	3.1x	7.4x	2.5x
3-d Gauss-Seidel	1.6x	1.1x	4.5x	1.5x
LU decomposition	5.6x	5.7x	14x	3.8x
Matrix Vec Transpose	9.3x	5.5x	13x	7x

- High speedups due to simultaneous optimization for parallelism and locality

Summary of performance improvement

Table: Improvement over state-of-the-art research compiler frameworks and native production compiler

Benchmark	Single core improvement		Multi-core speedup (4 cores)	
	over native compiler	over state-of-the-art research	over native compiler	over state-of-the-art research
Jacobi stencil (imperfect)	5.23x	2.1x	20x	2.7x
2-d FDTD	3.7x	3.1x	7.4x	2.5x
3-d Gauss-Seidel	1.6x	1.1x	4.5x	1.5x
LU decomposition	5.6x	5.7x	14x	3.8x
Matrix Vec Transpose	9.3x	5.5x	13x	7x

- High speedups due to simultaneous optimization for parallelism and locality

Related work

- Previous cost functions developed were for restrictive cases (like single perfectly-nested loops)
- Schedule-based approaches [Feautrier92, Darte/Vivien95, Griebl04 habilitation thesis]
- Affine partitioning [Lim/Lam PoPL'97, ICS'01] - minimizing order of synchronization is not sufficient; Ahmed/Pingali [IJPP'01] (heuristic scalability/practicality)
- Semi-automatic - URUK/WRAP-IT [Cohen05ICS, Girbal06IJPP] - very powerful flexible application of transformations specified manually by an expert

Related work

- Previous cost functions developed were for restrictive cases (like single perfectly-nested loops)
- Schedule-based approaches [Feautrier92, Darte/Vivien95, Griebl04 habilitation thesis]
- Affine partitioning [Lim/Lam PoPL'97, ICS'01] - minimizing order of synchronization is not sufficient; Ahmed/Pingali [IJPP'01] (heuristic scalability/practicality)
- Semi-automatic - URUK/WRAP-IT [Cohen05ICS, Girbal06IJPP] - very powerful flexible application of transformations specified manually by an expert

Related work

- Previous cost functions developed were for restrictive cases (like single perfectly-nested loops)
- Schedule-based approaches [Feautrier92, Darte/Vivien95, Griebl04 habilitation thesis]
- Affine partitioning [Lim/Lam PoPL'97, ICS'01] - minimizing order of synchronization is not sufficient; Ahmed/Pingali [IJPP'01] (heuristic scalability/practicality)
- Semi-automatic - URUK/WRAP-IT [Cohen05ICS, Girbal06IJPP] - very powerful flexible application of transformations specified manually by an expert

Conclusions

- Automatically finding good transforms for imperfectly nested loop sequences for coarse-grained parallelism and locality as is needed in practice
- A beta release of PLuTo (0.0.1) is available

  http://pluto-compiler.sourceforge.net

Conclusions

- Automatically finding good transforms for imperfectly nested loop sequences for coarse-grained parallelism and locality as is needed in practice
- A beta release of PLuTo (0.0.1) is available

  http://pluto-compiler.sourceforge.net

Future work

- Fusion/parallelization trade-off (which dependences between strongly-connected components to include/cut), interactions of fusion with tiling and prefetching
- Combine with stronger cost models for tile size selection
- Conservative dependence polyhedra for non-affine programs

Future work

- Fusion/parallelization trade-off (which dependences between strongly-connected components to include/cut), interactions of fusion with tiling and prefetching
- Combine with stronger cost models for tile size selection
- Conservative dependence polyhedra for non-affine programs

Acknowledgments

- Cédric Bastoul (Université Paris-Sud 11, INRIA Futurs) for **CLooG**, Paul Feautrier and all other contributors of CLooG and PipLib
- Martin Griebl (FMI, Universität Passau, Germany) and team for **LooPo**
- CC'08 reviewers

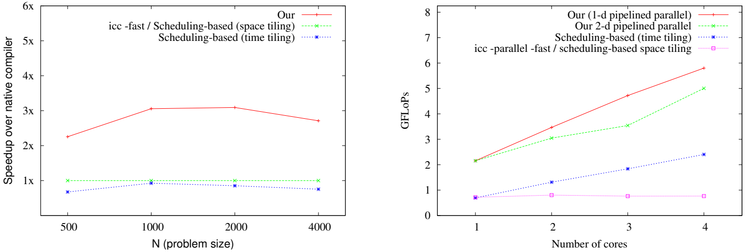
References

1. Automatic Transformations for Communication-Minimized Parallelization and Locality Optimization in the Polyhedral Model
   Uday Bondhugula, M. Baskaran, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan.
   OSU CISRC-TR43
2. A Practical and Fully Automatic Polyhedral Parallelizer and Locality Optimizer. Uday Bondhugula, A. Hartono, J. Ramanujam, and P. Sadayappan.
   ACM SIGPLAN PLDI'08, Jun 2008 (to appear).
   OSU-CISRC-TR70.

Thank you for your attention

- Questions?

2-d Finite Difference Time Domain



(a) Single core: T=500

(b) Parallel: $nx = ny = 2000$, $tmax = 500$

Figure: 2-d FDTD

Affine form of the Farkas Lemma

Let the hyperplanes bounding the polytope of statement S_k be given by:

$$a_{S,k} \begin{pmatrix} \vec{i} \\ \vec{n} \end{pmatrix} + b_{S,k} \geq 0, \quad k = 1, m_S$$

where $\vec{n}$ is a vector of the structure parameters. A well-known known result useful in the context of the polytope model is the affine form of the Farkas lemma.

Lemma (Affine form of Farkas Lemma)

Let $\mathcal{D}$ be a non-empty polyhedron defined by p affine inequalities or faces

$$a_k.x + b_k \geq 0, \quad k = 1, p$$

Then, an affine form ψ is non-negative everywhere in $\mathcal{D}$ iff it is a positive affine combination of the faces:

$$\psi(x) \equiv \lambda_0 + \sum_k \lambda_k (a_k x + b_k), \;\; \lambda \geq 0$$

Algorithm

Input Generalized dependence graph $G = (V, E)$ (includes dependence polyhedra P_e, $e \in E$)
S_{max}: statement with maximum domain dimensionality
for each dependence $e \in E$ **do**
Build legality constraints: apply Farkas Lemma on $\phi(\vec{t}) - \phi(f_e(\vec{t})) \geq 0$ under $\vec{t} \in P_e$, and eliminate all Farkas multipliers
Build communication volume/reuse distance bounding constraints: apply Farkas Lemma to $v(\vec{n}) - (\phi(\vec{t}) - \phi(f(\vec{t}))) \geq 0$ under $\vec{t} \in P_e$, and eliminate all Farkas multipliers
Aggregate constraints from both into $C_e(i)$
end for
repeat
$C = \emptyset$
for each dependence edge $e \in E$ **do**
$C \leftarrow C \cup C_e(i)$
end for
Compute lexicographic minimal solution with $u's$ coefficients in the leading position followed by w to iteratively find independent solutions to C (orthogonality constraints are added as each soln is found)
if no solutions were found **then**
Cut dependences between two strongly-connected components in the GDG and insert the appropriate *splitter* in the transformation matrices of the statements
end if
Compute E_c: dependences carried by solutions of Step 12/14; update necessary dependence polyhedra (when a portion of it is satisfied)
$E \leftarrow E - E_c$; reform the GDG (V, E)
until $H^{\perp}_{S_{max}} = \mathbf{0}$ and $E = \emptyset$
Output A transformation matrix for each statement (with the same number of rows)

Target architectures

- Transformation framework can be targeted towards general-purpose or special purpose multi-cores (current multicore processors)
- Different kinds/levels, and number of degrees of parallelism required

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

```
for (t=0; t<tmax; t++) {
  for (j=0; j<ny; j++)
    ey [0][ j] = exp(-coeff0*t1);

  for (i=1; i<nx; i++)
    for (j=0; j<ny; j++)
      ey[i][j] = ey[i][j] -
          coeff1*(hz[i][j]-hz[i-1][j]);

  for (i=0; i<nx; i++)
    for (j=1; j<ny; j++)
      ex[i][j] = ex[i][j]
      - coeff1*(hz[i][j]-hz[i][j-1]);

  for (i=0; i<nx; i++)
    for (j=0; j<ny; j++)
      hz[i][j] = hz[i][j] -
        coeff2*(ex[i][j+1]-ex[i][j]
              +ey[i+1][j]-ey[i][j]);
}
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{bmatrix}$$

Figure: 2-d Finite Difference Time Domain code