Lukas Truemper, Philipp Schaad, Berke Ates, Alexandru Calotoiu, Marcin Copik, Torsten Hoefler

A Priori Loop Nest Normalization: Automatic Loop Scheduling in Complex Applications

What is Loop Scheduling

HPC
Application

Focus on Loop Nests

```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

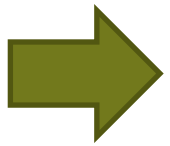
What is Loop Scheduling

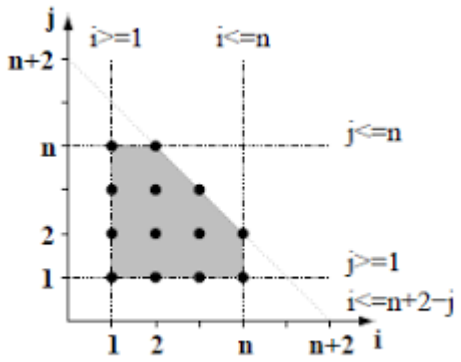
HPC
Application

Focus on Loop Nests

```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

Lifting





Representation, e.g.,
polyhedral model
stateful dataflow multigraphs

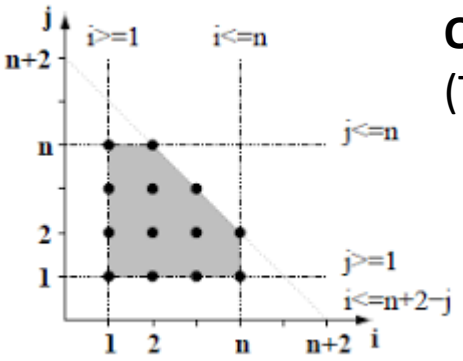
What is Loop Scheduling

HPC
Application

Focus on Loop Nests

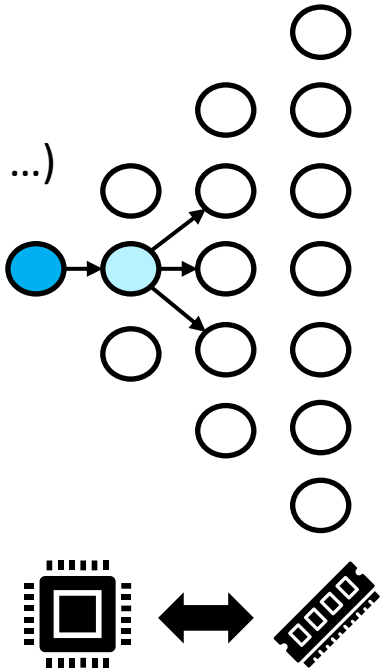
```
for (int i=1; i < 10000; i++)
    A[i] = B[i-1] + B[i] + B[i+1];
```

Lifting



Representation, e.g.,
polyhedral model
stateful dataflow multigraphs

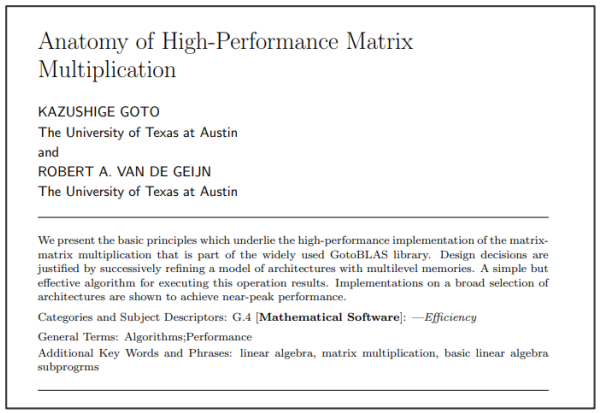
Optimization Space
(Tiling, Interchange, …)



Optimize
Performance

Loop Scheduling - State of Practice

Manual Tuning



Anatomy of High-Performance Matrix Multiplication

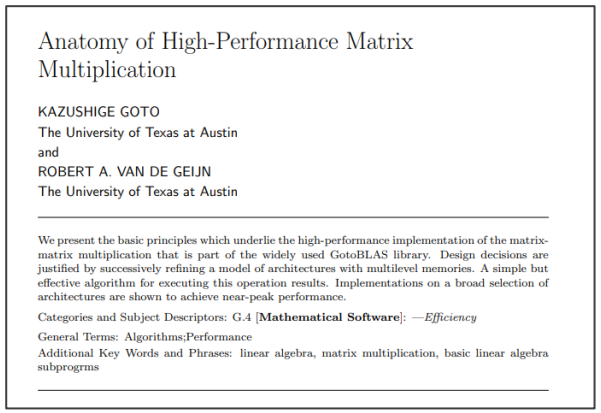
KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

We present the basic principles which underlie the high-performance implementation of the matrix-matrix multiplication that is part of the widely used GotoBLAS library. Design decisions are justified by successively refining a model of architectures with multilevel memories. A simple but effective algorithm for executing this operation results. Implementations on a broad selection of architectures are shown to achieve near-peak performance.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: —*Efficiency*
General Terms: Algorithms;Performance
Additional Key Words and Phrases: linear algebra, matrix multiplication, basic linear algebra subprogrms

Loop Scheduling - State of Practice

Manual Tuning



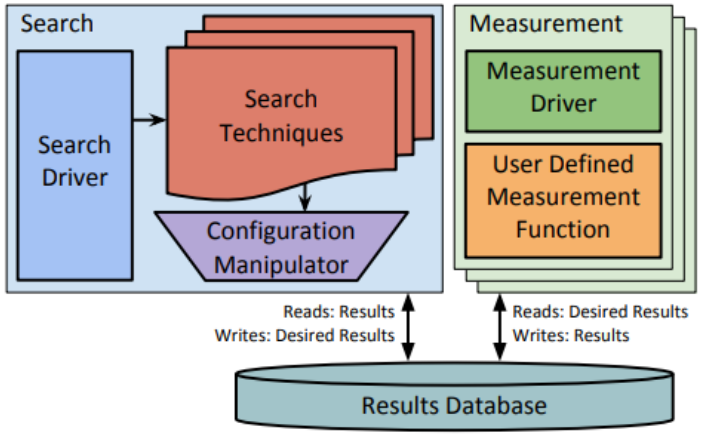
Anatomy of High-Performance Matrix Multiplication

KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

We present the basic principles which underlie the high-performance implementation of the matrix-matrix multiplication that is part of the widely used GotoBLAS library. Design decisions are justified by successively refining a model of architectures with multilevel memories. A simple but effective algorithm for executing this operation results. Implementations on a broad selection of architectures are shown to achieve near-peak performance.

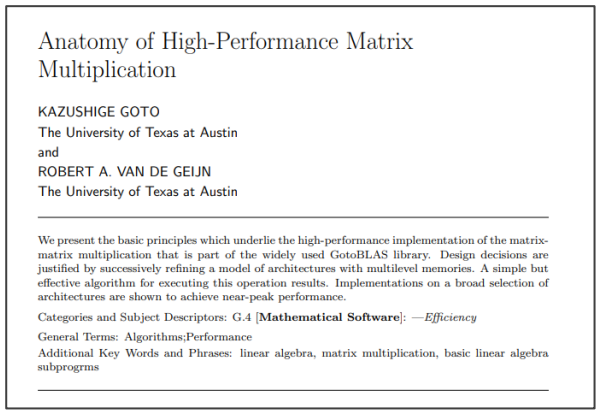
Categories and Subject Descriptors: G.4 [**Mathematical Software**]: —*Efficiency*
General Terms: Algorithms;Performance
Additional Key Words and Phrases: linear algebra, matrix multiplication, basic linear algebra subprogrms

Auto-tuner (Black-box Local Search)



Loop Scheduling - State of Practice

Manual Tuning



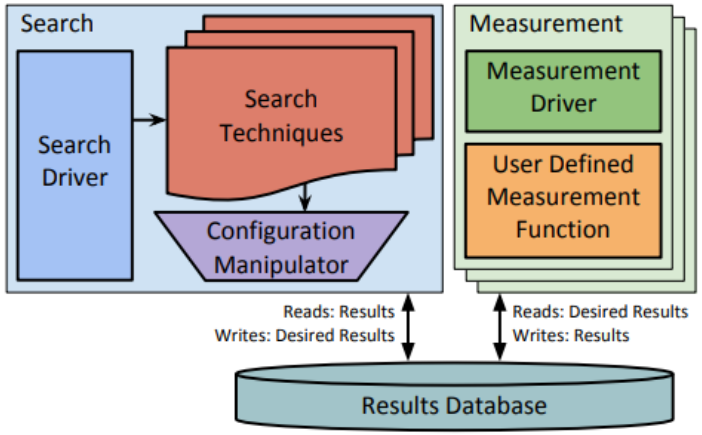
Anatomy of High-Performance Matrix Multiplication

KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

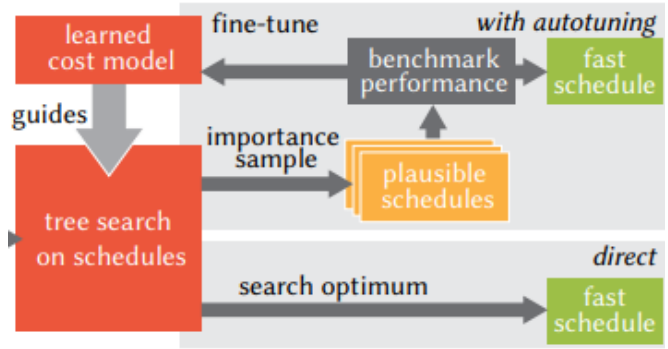
We present the basic principles which underlie the high-performance implementation of the matrix-matrix multiplication that is part of the widely used GotoBLAS library. Design decisions are justified by successively refining a model of architectures with multilevel memories. A simple but effective algorithm for executing this operation results. Implementations on a broad selection of architectures are shown to achieve near-peak performance.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: —*Efficiency*
General Terms: Algorithms;Performance
Additional Key Words and Phrases: linear algebra, matrix multiplication, basic linear algebra subprogrms

Auto-tuner (Black-box Local Search)

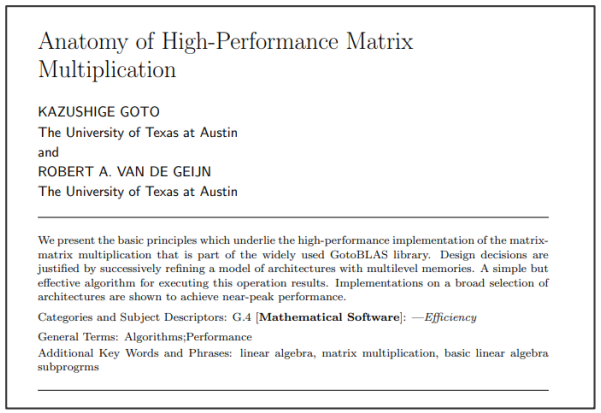


Auto-scheduler (Informed Local Search)



Loop Scheduling - State of Practice

Manual Tuning



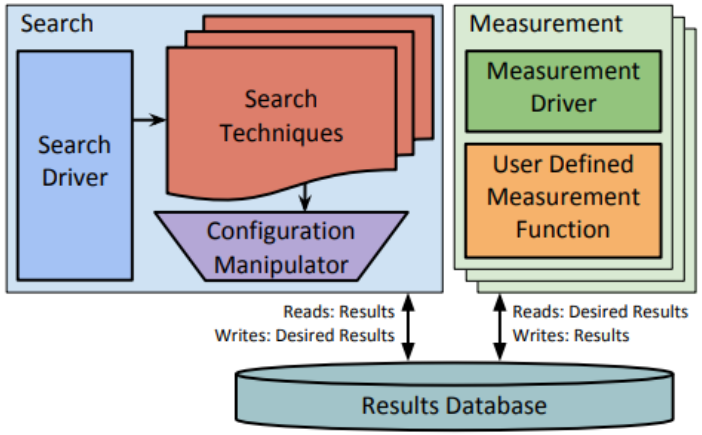
Anatomy of High-Performance Matrix Multiplication

KAZUSHIGE GOTO
The University of Texas at Austin
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

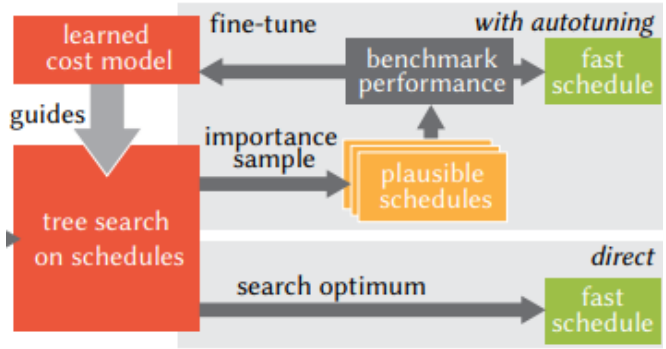
We present the basic principles which underlie the high-performance implementation of the matrix-matrix multiplication that is part of the widely used GotoBLAS library. Design decisions are justified by successively refining a model of architectures with multilevel memories. A simple but effective algorithm for executing this operation results. Implementations on a broad selection of architectures are shown to achieve near-peak performance.

Categories and Subject Descriptors: G.4 [**Mathematical Software**]: —*Efficiency*
General Terms: Algorithms;Performance
Additional Key Words and Phrases: linear algebra, matrix multiplication, basic linear algebra subprogrms

Auto-tuner (Black-box Local Search)



Auto-scheduler (Informed Local Search)



Given a loop nest, find **a path in the optimization space**

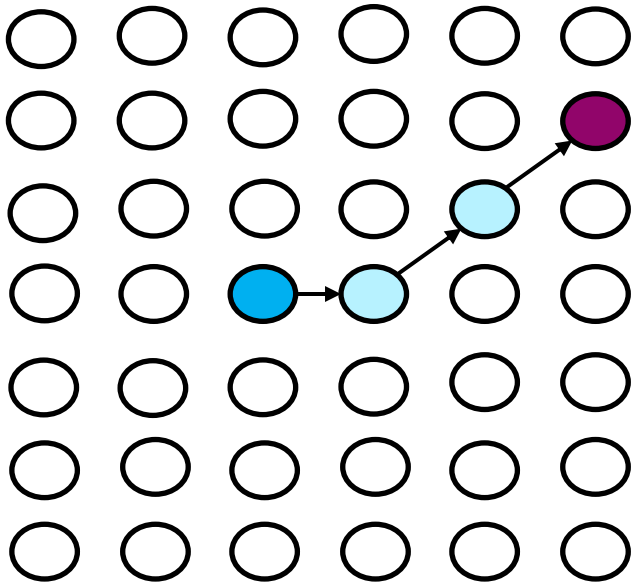
Loop Scheduling - Challenge

Initial state

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Target state

```
for (int k_tile = 0; k_tile < K; k_tile += 512) {
    for (int i_tile = 0; i_tile < N; i_tile += 32) {
        for (int j_tile = 0; j_tile < M; j_tile += 32) {
            for (int i = i_tile; i < i_tile + 32; i++) {
                for (int k = k_tile; k < k_tile + 512; k++) {
                    for (int j = j_tile; j < j_tile + 32; j++) {
                        C[i][j] += A[i][k] * B[k][j];
                    }
                }
            }
        }
    }
}
```



Optimization space with path

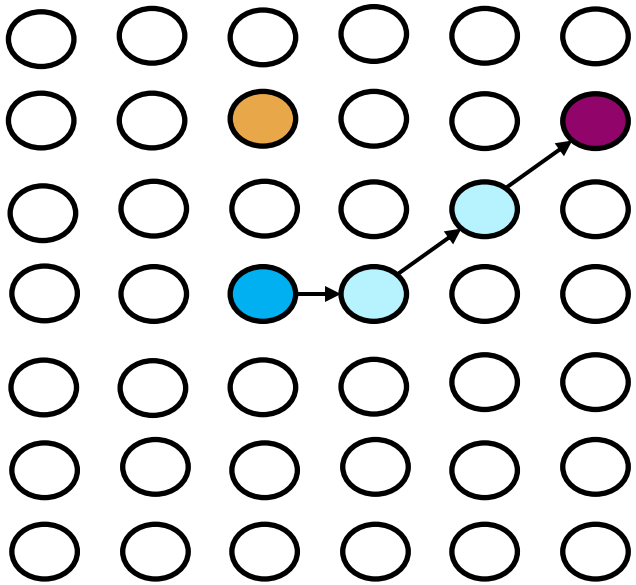
Loop Scheduling - Challenge

Initial state

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Algorithmic Variation

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```



Optimization space with path

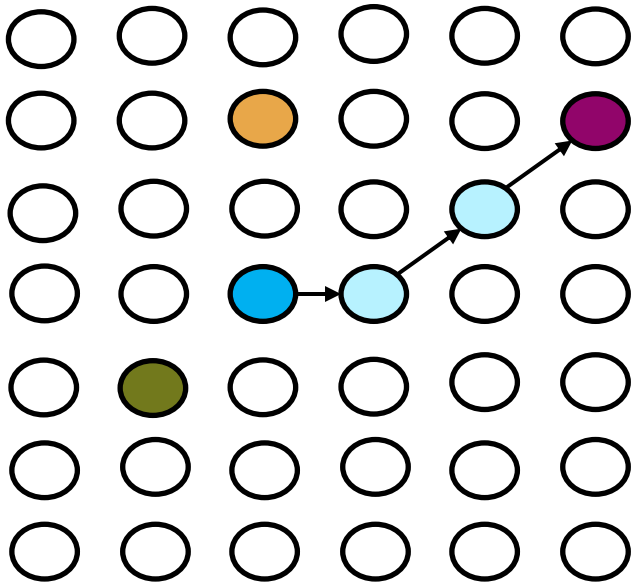
Loop Scheduling - Challenge

Initial state

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Language-specific Variation

```
C = A @ B
```



Optimization space with path

Loop Scheduling - Challenge

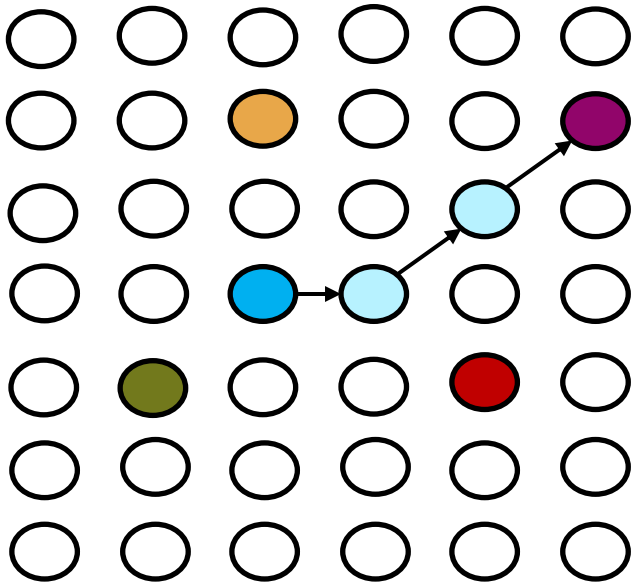
Initial state

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Pre-Optimized Code

```
for (int i_tile = 0; i_tile < N; i_tile += 8) {
    for (int j_tile = 0; j_tile < M; j_tile += 8) {
        for (int i = i_tile; i < i_tile + 8; i++) {
            for (int k = 0; k < K; k++) {
                for (int j = j_tile; j < j_tile + 8; j++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }
}
```

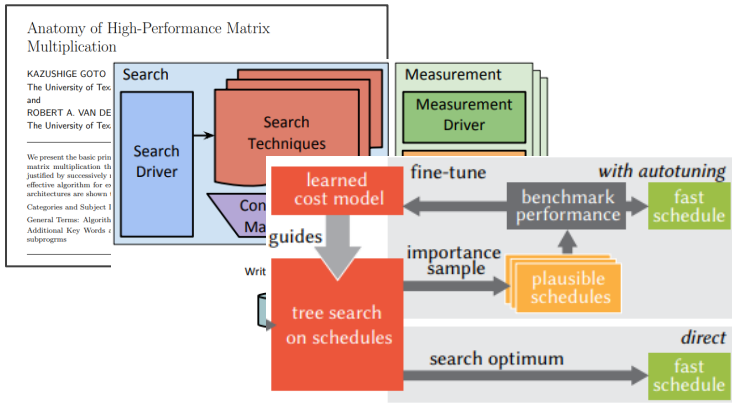
“Legacy Codes”



Optimization space with path

Loop Scheduling and the Generalization Problem

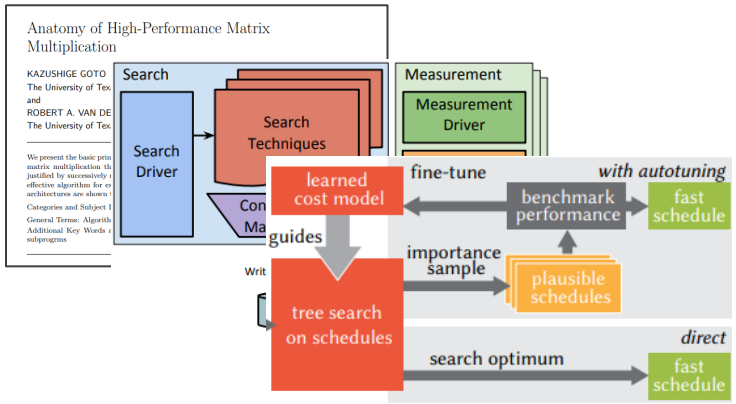
Local Search



Explore the paths of a limited set of problems

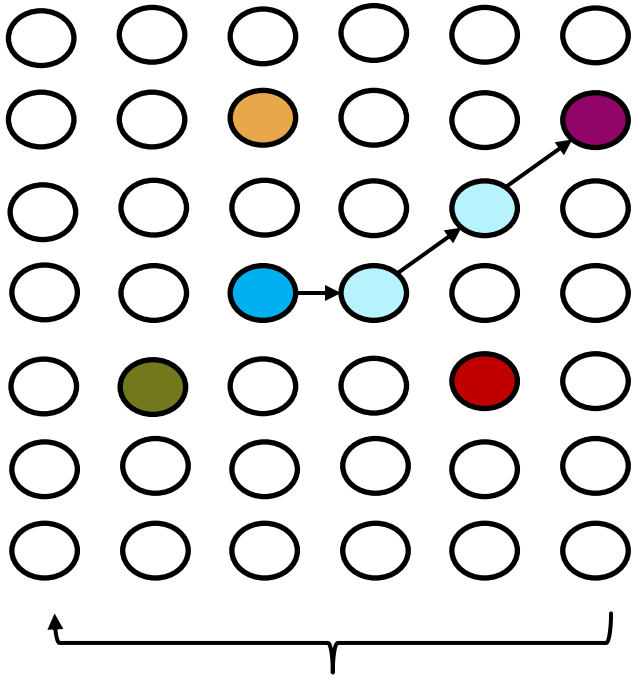
Loop Scheduling and the Generalization Problem

Local Search



Explore the paths of a limited set of problems

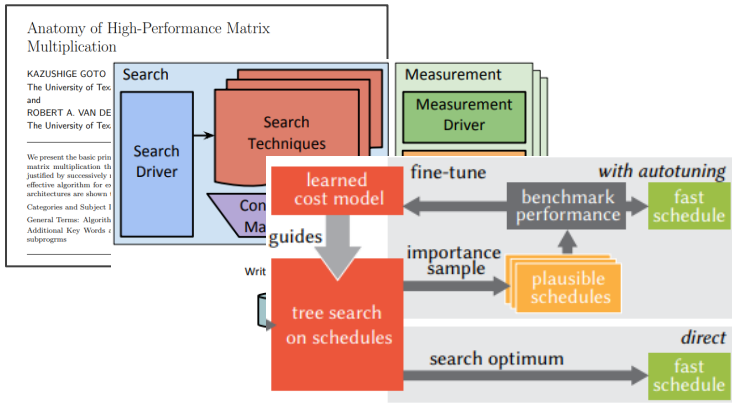
Optimization Space



Exponential variation of the same problem

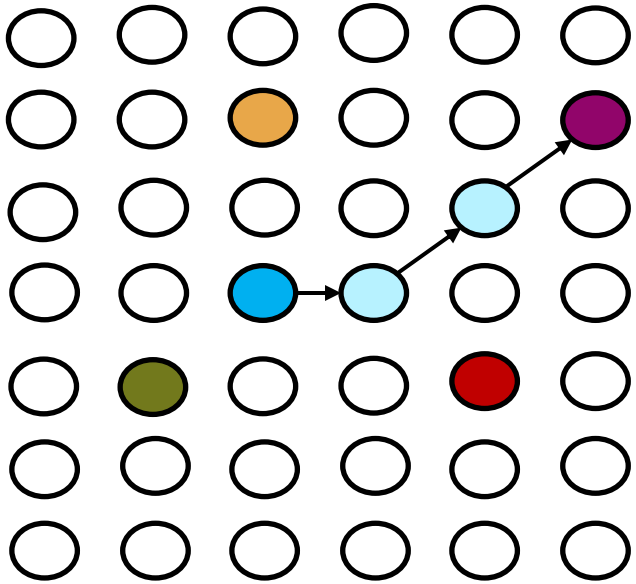
Loop Scheduling and the Generalization Problem

Local Search



Explore the paths of a limited set of problems

Optimization Space



Exponential variation of the same problem

"Generalization Problem"

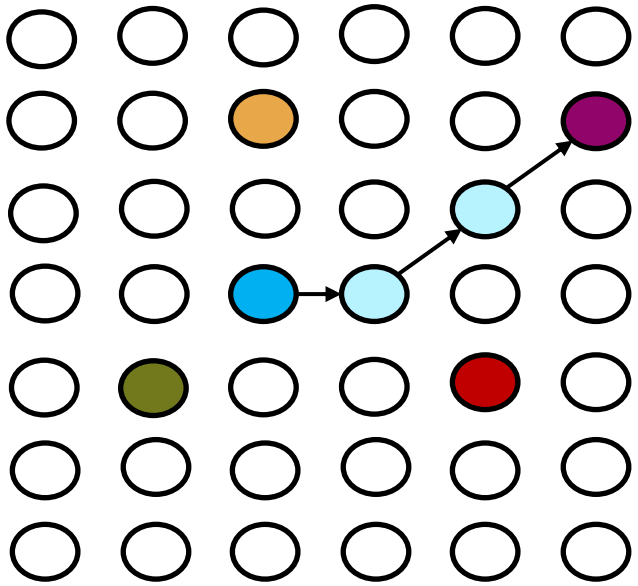
A Priori Loop Nest Normalization - Idea

Designated Initial State

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Algorithmic Variation

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```



Language-specific Variation

```
C = A @ B
```

Pre-Optimized Code

```
for (int i_tile = 0; i_tile < N; i_tile += 8) {
    for (int j_tile = 0; j_tile < M; j_tile += 8) {
        for (int i = i_tile; i < i_tile + 8; i++) {
            for (int k = 0; k < K; k++) {
                for (int j = j_tile; j < j_tile + 8; j++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }
}
```

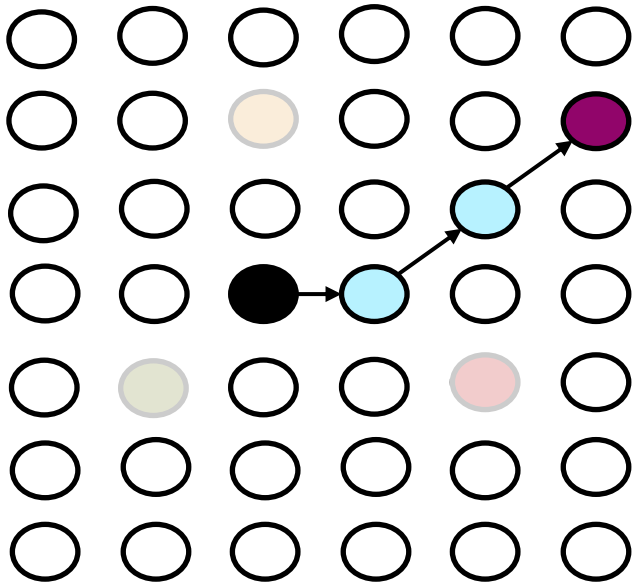
A Priori Loop Nest Normalization - Idea

Designated Initial State

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Language-specific Variation

```
C = A @ B
```



Algorithmic Variation

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```

Pre-Optimized Code

```
for (int i_tile = 0; i_tile < N; i_tile += 8) {
    for (int j_tile = 0; j_tile < M; j_tile += 8) {
        for (int i = i_tile; i < i_tile + 8; i++) {
            for (int k = 0; k < K; k++) {
                for (int j = j_tile; j < j_tile + 8; j++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }
}
```

Can we find a normalized initial state for the optimization?

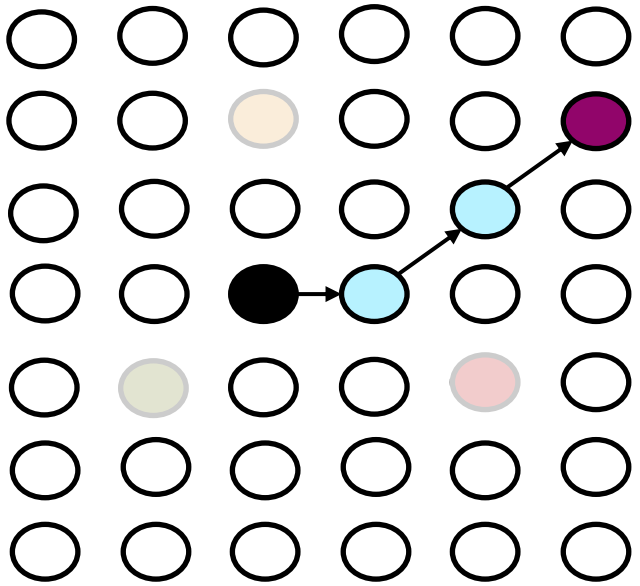
A Priori Loop Nest Normalization - Idea

Designated Initial State

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Language-specific Variation

```
C = A @ B
```



Algorithmic Variation

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```

Pre-Optimized Code

```
for (int i_tile = 0; i_tile < N; i_tile += 8) {
    for (int j_tile = 0; j_tile < M; j_tile += 8) {
        for (int i = i_tile; i < i_tile + 8; i++) {
            for (int k = 0; k < K; k++) {
                for (int j = j_tile; j < j_tile + 8; j++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }
}
```

Can we find a normalized initial state for the optimization?

<->

Loop nests must have same performance properties

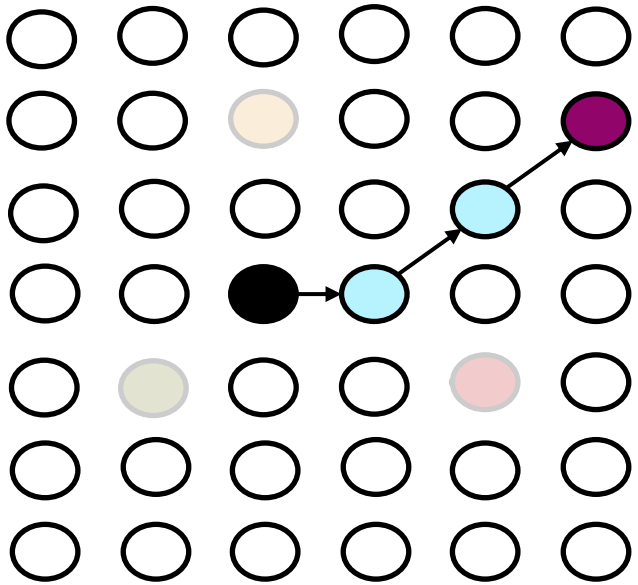
A Priori Loop Nest Normalization - Idea

Designated Initial State

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        for (int k = 0; k < K; k++) {
            C[i][j] += A[i][k] * B[k][j];
        }
    }
}
```

Language-specific Variation

```
C = A @ B
```



Algorithmic Variation

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        double sum = 0;
        for (int k = 0; k < K; k++) {
            sum += A[i][k] * B[k][j];
        }
        C[i][j] = sum;
    }
}
```

Pre-Optimized Code

```
for (int i_tile = 0; i_tile < N; i_tile += 8) {
    for (int j_tile = 0; j_tile < M; j_tile += 8) {
        for (int i = i_tile; i < i_tile + 8; i++) {
            for (int k = 0; k < K; k++) {
                for (int j = j_tile; j < j_tile + 8; j++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
    }
}
```

Can we find a normalized initial state for the optimization?

<->

Loop nests must have same performance properties

map to the same
memory access pattern

Normalization Criteria - Intuition

Matmul A (naive)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul B (k swap)

```
for (i = 0; i < N; i++) {
  for (k = 0; k < K; k++) {
    for (j = 0; j < M; j++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul C (with init)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    double sum = 0;
    for (k = 0; k < K; k++) {
      sum += A[i][k] * B[k][j];
    }
    C[i][j] = sum;
  }
}
```

Normalization Criteria - Intuition

Matmul A (naive)

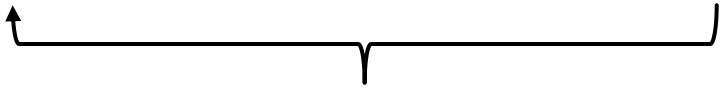
```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul B (k swap)

```
for (i = 0; i < N; i++) {
  for (k = 0; k < K; k++) {
    for (j = 0; j < M; j++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul C (with init)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    double sum = 0;
    for (k = 0; k < K; k++) {
      sum += A[i][k] * B[k][j];
    }
    C[i][j] = sum;
  }
}
```



Determine a normalized loop order

stride minimization

Normalization Criteria - Intuition

Matmul A (naive)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    for (k = 0; k < K; k++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul B (k swap)

```
for (i = 0; i < N; i++) {
  for (k = 0; k < K; k++) {
    for (j = 0; j < M; j++) {
      C[i][j] += A[i][k] * B[k][j];
    }
  }
}
```

Matmul C (with init)

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    double sum = 0;
    for (k = 0; k < K; k++) {
      sum += A[i][k] * B[k][j];
    }
```

Determine a normalized loop order

stride minimization

```
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++)
    SUM[i][j] = 0;
}
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++) {
    for (k = 0; k < K; k++)
      SUM[i][j] += A[i][k] * B[k][j];
  }
}
for (i = 0; i < N; i++) {
  for (j = 0; j < M; j++)
    C[i][j] = SUM[i][j];
}
```

Fission into “atomic” loop nests

maximal loop fission

A/B Testing for Loop Scheduling

PolyBench's GEMM

```
for (i = 0; i < _PB_NI; i++) {
  for (j = 0; j < _PB_NJ; j++)
C[i][j] *= beta;
  for (k = 0; k < _PB_NK; k++) {
    for (j = 0; j < _PB_NJ; j++)
  C[i][j] += alpha * A[i][k] * B[k][j];
  }
}
```

Not PolyBench's GEMM (j,k swapped)

```
for (i = 0; i < _PB_NI; i++) {
  for (j = 0; j < _PB_NJ; j++) {
    C[i][j] *= beta;
  }
  for (j = 0; j < _PB_NJ; j++) {
    for (k = 0; k < _PB_NK; k++) {
      C[i][j] += alpha * A[i][k] * B[k][j];
    }
  }
}
```

"A" implementation

"B" implementation

How robust are SOTA loop schedulers?

A/B Testing for Loop Scheduling

“Local” Loop Schedulers

1. ICC (Idiom Detection)

$$C(i,j) = \beta \cdot C(i,j) + \alpha \cdot \sum_{k=1}^{D} A(i,k) \cdot B(k,j).$$

A/B Testing for Loop Scheduling

"Local" Loop Schedulers

1. ICC (Idiom Detection)

$$C(i,j) = \beta \cdot C(i,j) + \alpha \cdot \sum_{k=1}^{D} A(i,k) \cdot B(k,j).$$

2. Polly (Integer Linear Programming / Pluto)

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{Z}^n}{\text{maximize}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

A/B Testing for Loop Scheduling

“Local” Loop Schedulers

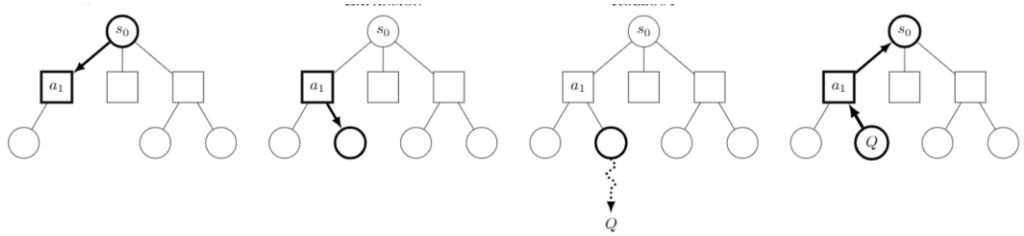
1. ICC (Idiom Detection)

$$C(i,j) = \beta \cdot C(i,j) + \alpha \cdot \sum_{k=1}^{D} A(i,k) \cdot B(k,j).$$

2. Polly (Integer Linear Programming / Pluto)

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{Z}^n}{\text{maximize}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

3. Tiramisu “Adapter” (Monte-Carlo Tree Search + NN)



A/B Testing for Loop Scheduling

“Local” Loop Schedulers

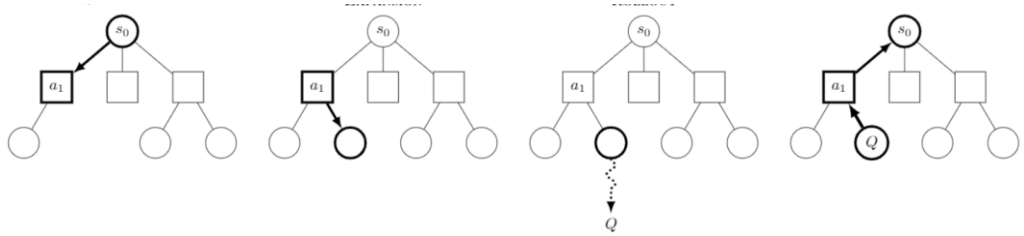
1. ICC (Idiom Detection)

$$C(i,j) = \beta \cdot C(i,j) + \alpha \cdot \sum_{k=1}^{D} A(i,k) \cdot B(k,j).$$

2. Polly (Integer Linear Programming / Pluto)

$$\begin{aligned} \underset{\mathbf{x}\in\mathbb{Z}^n}{\text{maximize}} \quad & \mathbf{c}^{\mathrm{T}}\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \leq \mathbf{b}, \\ & \mathbf{x} \geq \mathbf{0} \end{aligned}$$

3. Tiramisu “Adapter” (Monte-Carlo Tree Search + NN)



4. daisy (Normalization + Transfer Tuning)

“A” benchmarks → A Priori Loop Nest Normalization → Tiramisu + Autotuning → Optimization Database

Tune B, iff normalization(A) == normalization(B)

A/B Testing for Loop Scheduling

Benchmark	ICC [s]	Polly [s]	Tiramisu [s]	daisy [s]
2mm	0.3067	0.3371	0.0403	
3mm	**0.0212**	0.4747	0.1639	
atax	0.0125	0.0019	0.0085	
bicg	0.0104	0.0071	0.0071	
correlation	0.4111	**0.0327**	-	
covariance	0.4191	**0.0333**	-	
fdtd-2d	0.2220	1.6676	-	
gemm	0.0435	0.7698	0.2925	
gemver	0.0035	0.0030	0.0034	
gesummv	**0.0003**	0.0009	0.0011	
heat-3d	0.2819	2.0196	-	
Jacobi-2d	0.1976	0.7551	-	
mvt	**0.0011**	0.0018	0.0017	
syr2k	0.0876	0.0953	-	
syrk	0.0462	0.0620	-	

Runtime of A benchmarks (lower is better)

A/B Testing for Loop Scheduling

Benchmark	ICC [s]	Polly [s]	Tiramisu [s]	daisy [s]
2mm	0.3067	0.3371	0.0403	**0.0259**
3mm	**0.0212**	0.4747	0.1639	0.0304
atax	0.0125	0.0019	0.0085	**0.0015**
bicg	0.0104	0.0071	0.0071	**0.0018**
correlation	0.4111	**0.0327**	-	2.2869
covariance	0.4191	**0.0333**	-	0.3540
fdtd-2d	0.2220	1.6676	-	**0.2030**
gemm	0.0435	0.7698	0.2925	**0.0230**
gemver	0.0035	0.0030	0.0034	**0.0021**
gesummv	**0.0003**	0.0009	0.0011	0.0010
heat-3d	0.2819	2.0196	-	**0.2799**
Jacobi-2d	0.1976	0.7551	-	**0.1786**
mvt	**0.0011**	0.0018	0.0017	0.0016
syr2k	0.0876	0.0953	-	**0.0236**
syrk	0.0462	0.0620	-	**0.0172**

Runtime of A benchmarks (lower is better)

A/B Testing for Loop Scheduling

Benchmark	ICC	Polly	Tiramisu	daisy
2mm				
3mm				
atax				
bicg				
correlation				
covariance				
fdtd-2d				
gemm				
gemver				
gesummv				
heat-3d				
Jacobi-2d				
mvt				
syr2k				
syrk				

Runtime Ratio of A / B (closer to 1.00 is better)

A/B Testing for Loop Scheduling

Benchmark	ICC	Polly	Tiramisu	daisy
2mm	7.46			
3mm	0.92			
atax	10.42			
bicg	1.00			
correlation	1.00			
covariance	1.01			
fdtd-2d	0.12			
gemm	0.09			
gemver	0.38			
gesummv	0.15			
heat-3d	0.10			
Jacobi-2d	0.19			
mvt	1.00			
syr2k	0.98			
syrk	1.08			

Runtime Ratio of A / B (closer to 1.00 is better)

A/B Testing for Loop Scheduling

Benchmark	ICC	Polly	Tiramisu	daisy
2mm	7.46	0.37		
3mm	0.92	0.65		
atax	10.42	0.95		
bicg	1.00	0.87		
correlation	1.00	0.93		
covariance	1.01	0.99		
fdtd-2d	0.12	0.06		
gemm	0.09	2.46		
gemver	0.38	1.07		
gesummv	0.15	1.00		
heat-3d	0.10	0.53		
Jacobi-2d	0.19	0.78		
mvt	1.00	0.95		
syr2k	0.98	1.62		
syrk	1.08	1.77		

Runtime Ratio of A / B (closer to 1.00 is better)

A/B Testing for Loop Scheduling

Benchmark	ICC	Polly	Tiramisu	daisy
2mm	7.46	0.37	0.08	
3mm	0.92	0.65	0.48	
atax	10.42	0.95	0.81	
bicg	1.00	0.87	0.89	
correlation	1.00	0.93	-	
covariance	1.01	0.99	-	
fdtd-2d	0.12	0.06	-	
gemm	0.09	2.46	2.32	
gemver	0.38	1.07	0.31	
gesummv	0.15	1.00	0.23	
heat-3d	0.10	0.53	-	
Jacobi-2d	0.19	0.78	-	
mvt	1.00	0.95	0.20	
syr2k	0.98	1.62	-	
syrk	1.08	1.77	-	

Runtime Ratio of A / B (closer to 1.00 is better)

A/B Testing for Loop Scheduling

Benchmark	ICC	Polly	Tiramisu	daisy
2mm	7.46	0.37	0.08	1.01
3mm	0.92	0.65	0.48	0.89
atax	10.42	0.95	0.81	0.94
bicg	1.00	0.87	0.89	1.13
correlation	1.00	0.93	-	0.99
covariance	1.01	0.99	-	1.00
fdtd-2d	0.12	0.06	-	1.00
gemm	0.09	2.46	2.32	1.11
gemver	0.38	1.07	0.31	1.05
gesummv	0.15	1.00	0.23	0.91
heat-3d	0.10	0.53	-	1.00
Jacobi-2d	0.19	0.78	-	1.02
mvt	1.00	0.95	0.20	1.07
syr2k	0.98	1.62	-	1.07
syrk	1.08	1.77	-	1.05

Runtime Ratio of A / B (closer to 1.00 is better)

A/B Testing for Loop Scheduling

Benchmark	ICC	Polly	Tiramisu	daisy
2mm	7.46	0.37	0.08	1.01
3mm	0.92	0.65	0.48	0.89
atax	10.42	0.95	0.81	0.94
bicg	1.00	0.87	0.89	1.13
correlation	1.00	0.93	-	0.99
covariance	1.01	0.99	-	1.00
fdtd-2d	0.12	0.06	-	1.00
gemm	0.09	2.46	2.32	1.11
gemver	0.38	1.07	0.31	1.05
gesummv	0.15	1.00	0.23	0.91
heat-3d	0.10	0.53	-	1.00
Jacobi-2d	0.19	0.78	-	1.02
mvt	1.00	0.95	0.20	1.07
syr2k	0.98	1.62	-	1.07
syrk	1.08	1.77	-	1.05

Runtime Ratio of A / B (closer to 1.00 is better)

Loop Scheduling Beyond Programming Languages – “C” Implementation

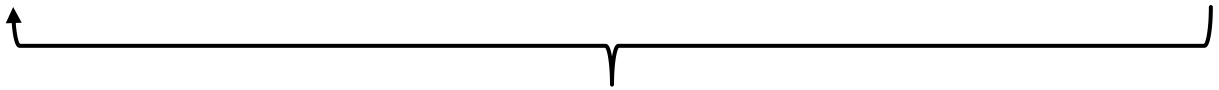
PolyBench’s SYRK (C)

```c
for (int i = 0; i < _PB_N; i++) {
  for (int j = 0; j <= i; j++)
    C[i][j] *= beta;
  for (int k = 0; k < _PB_M; k++)
    for (int j = 0; j < i; j++)
      C[i][j] += alpha * A[i][k] * B[j][k];
}
```

NPBench’s SYRK (Python)

```python
for i in range(A.shape[0]):
    C[i,:i+1] *= beta
    for k in range(A.shape[1]):
        C[i,:i+1] += alpha * A[i,k] * A[:i+1,k]
```

Different frontend, different representation



Can we use optimizations extracted from PolyBench to tune NPBench equivalents?

Case Study: CLOUDSC

Recipe:
Apply producer-consumer loop fusion to application (eliminate temporary arrays)

```
! Function Definitions
REAL :: FOEEWM, FOELDCPM
FOEEWM ( PTARE ) = R2ES *&
   &(MIN(1.0,((MAX(RTICE,MIN(RTWAT,PTARE))-RTICE)*RTWAT_RTICE_R)**2)&
   & *EXP(R3LES*(PTARE-RTT)/(PTARE-R4LES))&
   & +(1.0-MIN(1.0,((MAX(RTICE,MIN(RTWAT,PTARE))-RTICE)*RTWAT_RTICE_R)**2))&
   & *EXP(R3IES*(PTARE-RTT)/(PTARE-R4IES)))
FOELDCPM ( PTARE ) = MIN(1.0,((MAX(RTICE,MIN(RTWAT,PTARE))-&
   &RTICE)*RTWAT_RTICE_R)**2)*RALVDCP+&
   &(1.0-MIN(1.0,((MAX(RTICE,MIN(RTWAT,PTARE))-RTICE)*RTWAT_RTICE_R)**2))*RALSDCP

! Vertical loop
DO JK=NCLDTOP,KLEV
...
    ! Loop Nest
    DO JL=KIDIA,KFDIA
        ZQP   = 1.0/PAP(JL,JK)
        ZQSAT = FOEEWM(ZTP1(JL,JK))*ZQP
        ZQSAT = MIN(0.5,ZQSAT)
        ZCOR  = 1.0/(1.0-RETV  *ZQSAT)
        ZQSAT = ZQSAT*ZCOR
        ZCOND = (ZQSMIX(JL,JK)-ZQSAT)/(1.0+ZQSAT*ZCOR*FOEDEM(ZTP1(JL,JK)))
        ZTP1(JL,JK) = ZTP1(JL,JK)+FOELDCPM(ZTP1(JL,JK))*ZCOND
        ZQSMIX(JL,JK) = ZQSMIX(JL,JK)-ZCOND
        ZQSAT = FOEEWM(ZTP1(JL,JK))*ZQP
        ZQSAT = MIN(0.5,ZQSAT)
        ZCOR  = 1.0/(1.0-RETV  *ZQSAT)
        ZQSAT = ZQSAT*ZCOR
        ZCOND1= (ZQSMIX(JL,JK)-ZQSAT)/(1.0+ZQSAT*ZCOR*FOEDEM(ZTP1(JL,JK)))
        ZTP1(JL,JK) = ZTP1(JL,JK)+FOELDCPM(ZTP1(JL,JK))*ZCOND1
        ZQSMIX(JL,JK) = ZQSMIX(JL,JK)-ZCOND1
    ENDDO
...
ENDDO
```

One such loop nest:
aggressively hand-tuned for a processor many years ago

Case Study: CLOUDSC

Recipe + Normalization:
Apply producer-consumer loop fusion to application
(eliminate temporary arrays)

```
...
! Vertical loop
DO JK=NCLDTOP,KLEV
...
    ! Loop Nest
    DO JL=KIDIA,KFDIA
       ZQP_0(JL)   = 1.0/PAP(JL,JK)
    ENDDO
    DO JL=KIDIA,KFDIA
       ZQSAT = FOEEWM(ZTP1(JL,JK))*ZQP_0(JL)
       ZQSAT = MIN(0.5,ZQSAT)
       ZCOR  = 1.0/(1.0-RETV  *ZQSAT)
       ZQSAT = ZQSAT*ZCOR
       ZCOND_0(JL) = (ZQSMIX(JL,JK)-ZQSAT)/(1.0+ZQSAT*ZCOR*FOEDEM(ZTP1(JL,JK)))
    ENDDO
    DO JL=KIDIA,KFDIA
       ZTP1(JL,JK) = ZTP1(JL,JK)+FOELDCPM(ZTP1(JL,JK))*ZCOND_0(JL)
    ENDDO
    DO JL=KIDIA,KFDIA
       ZQSMIX(JL,JK) = ZQSMIX(JL,JK)-ZCOND_0(JL)
    ENDDO
    DO JL=KIDIA,KFDIA
       ZQSAT = FOEEWM(ZTP1(JL,JK))*ZQP_0(JL)
       ZQSAT = MIN(0.5,ZQSAT)
       ZCOR  = 1.0/(1.0-RETV  *ZQSAT)
       ZQSAT = ZQSAT_4(JL)*ZCOR
       ZCOND1_0(JL)= (ZQSMIX(JL,JK)-ZQSAT)/(1.0+ZQSAT*ZCOR*FOEDEM(ZTP1(JL,JK)))
    ENDDO
    DO JL=KIDIA,KFDIA
       ZTP1(JL,JK) = ZTP1(JL,JK)+FOELDCPM(ZTP1(JL,JK))*ZCOND1_0(JL)
    ENDDO
    DO JL=KIDIA,KFDIA
       ZQSMIX(JL,JK) = ZQSMIX(JL,JK)-ZCOND1_0(JL)
    ENDDO
...
ENDDO
```

Conclusion: A Priori Loop Nest Normalization

More of SPCL's research:

youtube.com/@spcl — 180+ Talks

twitter.com/spcl_eth — 1.2K+ Followers

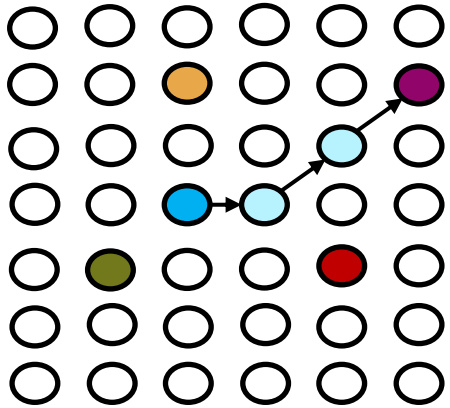
github.com/spcl — 2K+ Stars

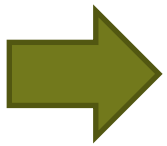
… or spcl.ethz.ch

Language-specific Variation

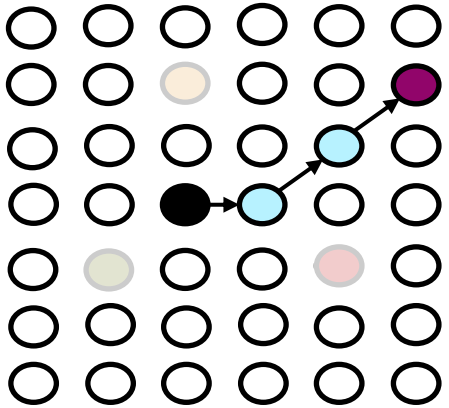
Algorithmic Variation

Pre-Optimized Code





Normalize loops' memory access pattern



Schedule exponential variations of loop nests for free

Conclusion: A Priori Loop Nest Normalization

More of SPCL's research:

youtube.com/@spcl — 180+ Talks

twitter.com/spcl_eth — 1.2K+ Followers

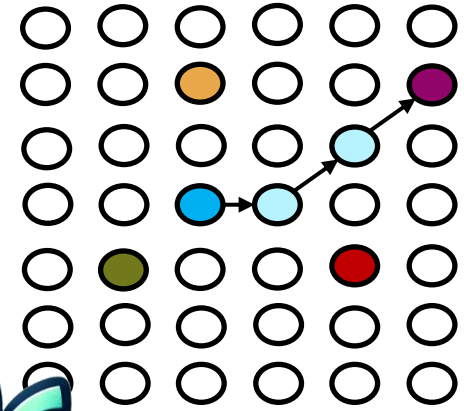
github.com/spcl — 2K+ Stars

… or spcl.ethz.ch

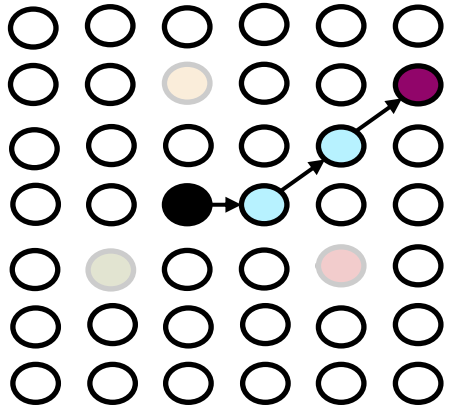
Language-specific Variation

Algorithmic Variation

Pre-Optimized Code



Normalize loops' memory access pattern



Daisytuner

Daisytuner.com

Hiring! Hackathons in April (Darmstadt)!

Schedule exponential variations of loop nests for free