A Language for the Compact Representation of Multiple Program Versions

Proceedings of the 18th International Workshop on Languages and Compilers for Parallel Computing (2005)

Sebastien Donadio[1,2], James Brodman[4], Thomas Roeder[5], Kamen Yotov[5], Denis Barthou[2], Albert Cohen[3], María Jesús Garzarán[4], David Padua[4], and Keshav Pingali[5]

[1] BULL SA
[2] University of Versailles St-Quentin-en-Yvelines
[3] INRIA Futurs
[4] University of Illinois at Urbana-Champaign
[5] Cornell University

Pascal Fischli, 9. November 2011

All examples are taken from this paper

Motivation

- **Wanted: Best Program Version**

Motivation

- **Wanted: Best Program Version**
- **Library Generators have Weaknesses**

Motivation

- **Wanted: Best Program Version**
- **Library Generators have Weaknesses**
  - Specification of Transformations
    - Which
    - Where
    - Order
    - How

Motivation

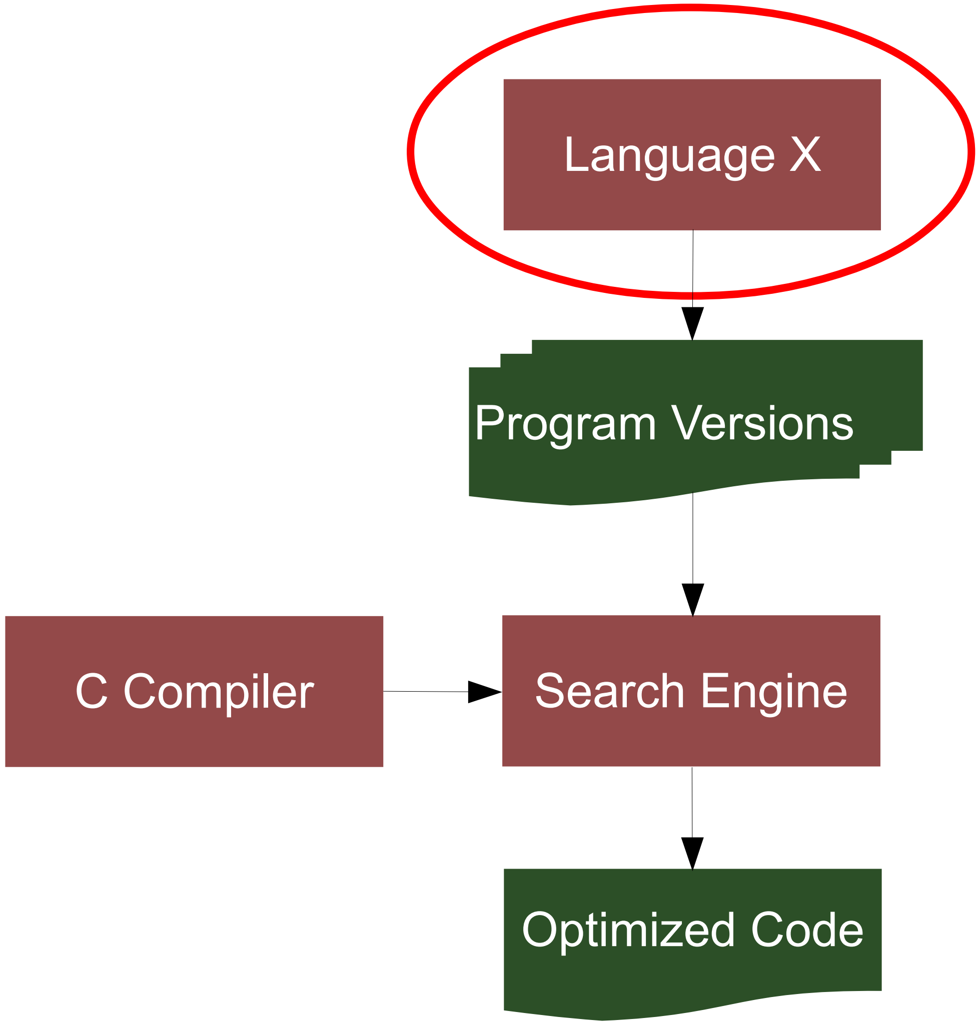
- **Wanted: Best Program Version**
- **Library Generators have Weaknesses**
  - Specification of Transformations
    - Which
    - Where
    - Order
    - How
  - Representation of Program Versions
    - Natural and Compact

Motivation

- **Wanted: Best Program Version**
- **Library Generators have Weaknesses**
  - Specification of Transformations
    - Which
    - Where
    - Order
    - How
  - Representation of Program Versions
    - Natural and Compact
  - Defining of new Transformations

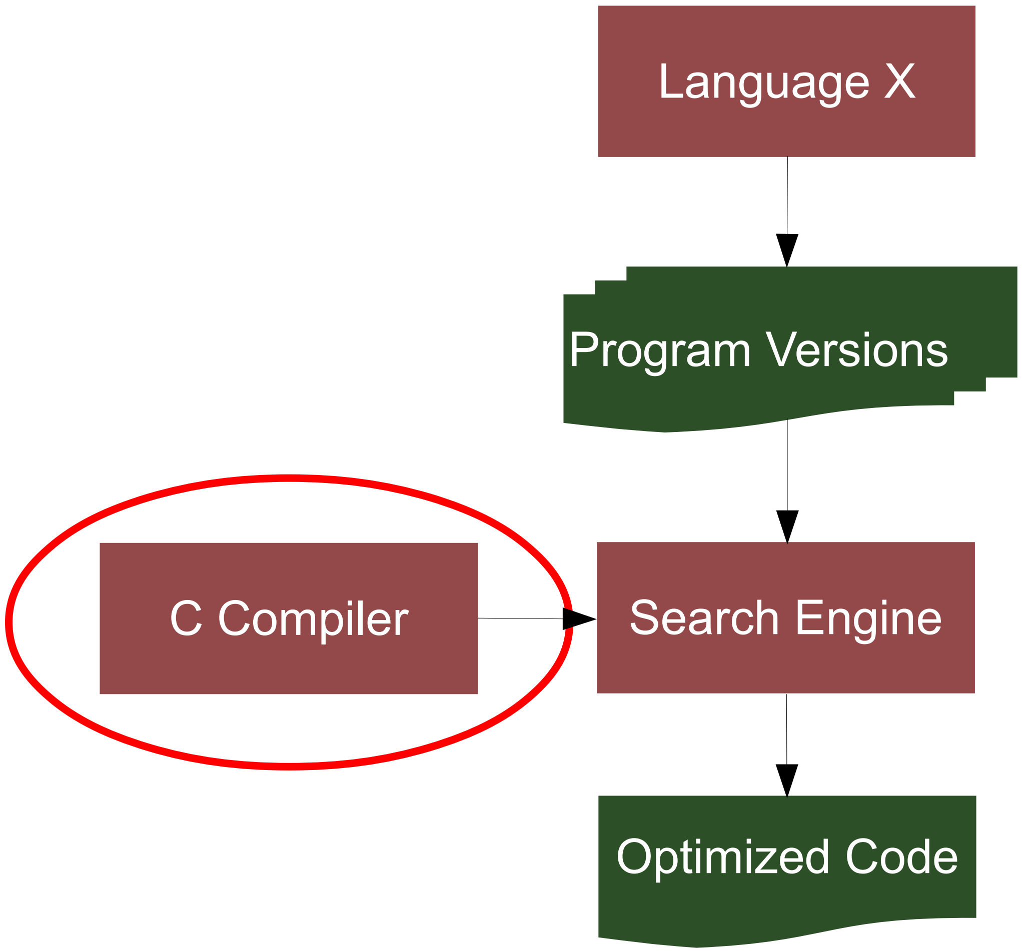
Language X - Workflow

- **Language Usages**
  - Write Programs in X directly
  - Intermediate Representation



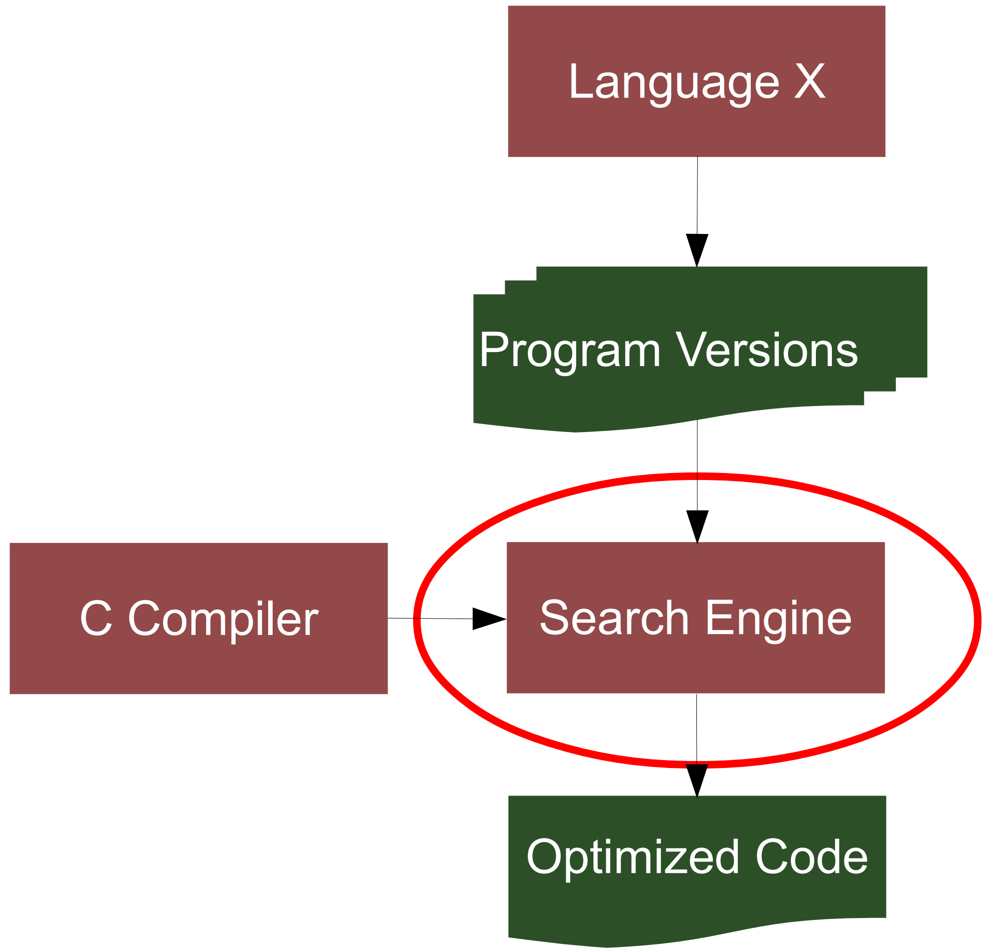
Language X - Workflow

- **Language Usages**
  - Write Programs in X directly
  - Intermediate Representation
- **Native C Compilers**
  - Low-Level Optimizations
  - May undo Transformations in X



Language X - Workflow

- **Language Usages**
  - Write Programs in X directly
  - Intermediate Representation
- **Native C Compilers**
  - Low-Level Optimizations
  - May undo Transformations in X
- **Search Engine**
  - Exhaustive Search
  - Parameter Values



Transformations – Important Features

- **Elementary Transformations**
  - Sequences of Statements
  - Loops

Transformations – Important Features

- **Elementary Transformations**
  - Sequences of Statements
  - Loops
- **Composition of Transformations**
  - Conditional

Transformations – Important Features

- **Elementary Transformations**
  - Sequences of Statements
  - Loops
- **Composition of Transformations**
  - Conditional
- **Mechanism to name Statements**

Transformations – Important Features

- **Elementary Transformations**
  - Sequences of Statements
  - Loops
- **Composition of Transformations**
  - Conditional
- **Mechanism to name Statements**
- **Procedural Abstraction**

Transformations – Important Features

- **Elementary Transformations**
  - Sequences of Statements
  - Loops
- **Composition of Transformations**
  - Conditional
- **Mechanism to name Statements**
- **Procedural Abstraction**
- **Mechanism to define new Transformations**

Macros as Language Representation

- **Simple Example**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

Macros as Language Representation

- **Simple Example**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

- **X Representation**

```
sum = 0;
for (i=0;i<256;i+=%d) {
    %for (k=0; k<=(%d-1); k++)
        s = s + a[i+%k];
}
```

Macros as Language Representation

- **Simple Example**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

- **X Representation**

```
sum = 0;
for (i=0;i<256;i+=%d) {
    %for (k=0; k<=(%d-1); k++)
        s = s + a[i+%k];
}
```

- **Which stands for**

```
sum = 0;
for (i=0;i<256;i+=%d) {
    s = s + a[i];
    s = s + a[i+1];
    ...
    s = s + a[i+(%d-1)];
}
```

Macros as Language Representation

- **Simple Example**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

- **X Representation**

```
sum = 0;
for (i=0;i<256;i+=%d) {
    %for (k=0; k<=(%d-1); k++)
        s = s + a[i+%k];
}
```

- **Which stands for**

```
sum = 0;
for (i=0;i<256;i+=%d) {
    s = s + a[i];
    s = s + a[i+1];
    ...
    s = s + a[i+(%d-1)];
}
```

Seems complicated?

Macros again: Tiled MMM-Loop

```
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];
}}}
```

```
for (i=0;i<(N/%tile)*%tile;i+=%tile) {
   for (j=0;j<(M/%tile)*%tile;j+=%tile) {
      for (k=0;k<(K/%tile)*%tile;k+=%tile) {
         for (ii=i;ii<i+%tile;i++) {
            for (jj=j;jj<j+%tile;j++) {
               for (kk=k;kk<k+%tile;kk++) {
                  c[ii][jj] += a[ii][kk] * b[kk][jj];
      }}}}
      %if ((K/%tile)*%tile)!=K) {
         for (k=(K/%tile)*%tile;k<;k++) {
            for (ii=i;ii<i+%tile;i++) {
               for (jj=j;jj<j+%tile;j++) {
                  for (kk=k;kk<k+%tile;kk++) {
                     c[ii][jj] += a[ii][kk] * b[kk][jj];
}}}}}}
....
```

Better Representation: Pragmas

- **Begin/End**

```
#pragma xlang begin
.
.
.
#pragma xlang end
```

- **Naming**
  - {} for set of statements

```
#pragma xlang name <id> {...}
```

- **Transformation**
  - Basic Syntax

```
#pragma xlang transform keyword <list-input-par> <list-output-par>
```

Implemented Elementary Transformations

- **Full Unrolling**
- **Partial Unrolling**
- **Strip Mining**
- **Interchange**
- **Loop Fission**
- **Loop Fusion**
- **Scalar Promote**
- **Lifting**
- **Sofware Pipelining**

Example 1: Loop Unroll

- **Once again the simple Loop**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

Example 1: Loop Unroll

- **Once again the simple Loop**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

- **X Representation**

```
sum=0;
#pragma xlang name l1
for (i=0;i<256;i++) {
    s = s + a[i];
}
#pragma xlang transform unroll l1 4
```

Example 1: Loop Unroll

- **Once again the simple Loop**

```
sum = 0;
for (i=0;i<256;i++) {
    s = s + a[i];
}
```

- **X Representation**

```
sum=0;
#pragma xlang name l1
for (i=0;i<256;i++) {
    s = s + a[i];
}
#pragma xlang transform unroll l1 4
```

- **Resulting Code**

```
sum=0;
#pragma xlang name l1
for (i=0;i<256;i+=4) {
    s = s + a[i];
    s = s + a[i+1];
    s = s + a[i+2];
    s = s + a[i+3];
}
```

Example 2: Pipelining

- **The MMM-Loop again**

```
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];
}}}
```

Example 2: Pipelining

- **The MMM-Loop again**

```
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];
}}}
```

```
for (i=0;i<N;i++){
    for (j=0;j<M;j++) {
        for (k=0;k<K;k++) {
            #pragma xlang name statement st1
            c[i][j] += a[i][k] * b[k][j];
}}}
#pragma xlang transform split st1 st2 temp
```

- **X Representation**

Example 2: Pipelining

- **The MMM-Loop again**

```
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];
}}}
```

- **X Representation**

```
for (i=0;i<N;i++){
    for (j=0;j<M;j++) {
        for (k=0;k<K;k++) {
            #pragma xlang name statement st1
            c[i][j] += a[i][k] * b[k][j];
}}}
#pragma xlang transform split st1 st2 temp
```

- **Resulting Code**

```
double temp[0..K];
for (i=0;i<N;i++){
    for (j=0;j<M;j++) {
        for (k=0;k<K;k++) {
            #pragma xlang name statement st1
            temp[k] = a[i][k] * b[k][j];
            #pragma xlang name statement st2
            c[i][j] = c[i][j] + temp[k];
}}}
```

Defining of new Transformations

- **Pattern Rewriting**
  - 1. Pattern: Matching
  - 2. Pattern: Rewriting
- **Macro Code directly**

Experimental Results

- **Matrix-Matrix Multiplication (DGEMM)**
- **Mimic ATLAS**
- **Focus on Blocking for L2 and L3 cache**
- **Compiler Intel C compiler (icc) 8.1**
  - Pipelining
  - Block Scheduling

Experimental Results – X Code

```
#pragma xlang name iloop
for (i=0;i<NB;i++)
    #pragma xlang name jloop
    for (j=0;j<NB;j++)
        #pragma xlang name kloop
        for (k=0;k<NB;k++) {
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
#pragma xlang transform stripmine iloop NU NUloop
#pragma xlang transform stripmine jloop MU MUloop
#pragma xlang transform interchange kloop MUloop
#pragma xlang transform interchange jloop NUloop
#pragma xlang transform interchange kloop NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform scalarize_in b in kloop
#pragma xlang transform scalarize_in a in kloop
#pragma xlang transform scalarize_in&out c in kloop
#pragma xlang transform lift kloop.loads before kloop
#pragma xlang transform lift kloop.stores after kloop
```

Experimental Results – X Code

```
#pragma xlang name iloop
for (i=0;i<NB;i++)
    #pragma xlang name jloop
    for (j=0;j<NB;j++)
        #pragma xlang name kloop
        for (k=0;k<NB;k++) {
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
#pragma xlang transform stripmine iloop NU NUloop
#pragma xlang transform stripmine jloop MU MUloop
#pragma xlang transform interchange kloop MUloop
#pragma xlang transform interchange jloop NUloop
#pragma xlang transform interchange kloop NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform scalarize_in b in kloop
#pragma xlang transform scalarize_in a in kloop
#pragma xlang transform scalarize_in&out c in kloop
#pragma xlang transform lift kloop.loads before kloop
#pragma xlang transform lift kloop.stores after kloop
```

Tiling iloop and jloop

Experimental Results – X Code

```
#pragma xlang name iloop
for (i=0;i<NB;i++)
    #pragma xlang name jloop
    for (j=0;j<NB;j++)
        #pragma xlang name kloop
        for (k=0;k<NB;k++) {
            c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
#pragma xlang transform stripmine iloop NU NUloop
#pragma xlang transform stripmine jloop MU MUloop
#pragma xlang transform interchange kloop MUloop
#pragma xlang transform interchange jloop NUloop
#pragma xlang transform interchange kloop NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform scalarize_in b in kloop
#pragma xlang transform scalarize_in a in kloop
#pragma xlang transform scalarize_in&out c in kloop
#pragma xlang transform lift kloop.loads before kloop
#pragma xlang transform lift kloop.stores after kloop
```

NU = 1
MU = 4

Tiling iloop and jloop

Experimental Results – X Code

```
#pragma xlang name iloop
for (i=0;i<NB;i++)
   #pragma xlang name jloop
   for (j=0;j<NB;j++)
      #pragma xlang name kloop
      for (k=0;k<NB;k++) {
         c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
#pragma xlang transform stripmine iloop NU NUloop
#pragma xlang transform stripmine jloop MU MUloop
#pragma xlang transform interchange kloop MUloop
#pragma xlang transform interchange jloop NUloop
#pragma xlang transform interchange kloop NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform scalarize_in b in kloop
#pragma xlang transform scalarize_in a in kloop
#pragma xlang transform scalarize_in&out c in kloop
#pragma xlang transform lift kloop.loads before kloop
#pragma xlang transform lift kloop.stores after kloop
```

NU = 1
MU = 4

Tiling iloop and jloop

Full Unroll the new tiles

Experimental Results – X Code

```
#pragma xlang name iloop
for (i=0;i<NB;i++)
   #pragma xlang name jloop
   for (j=0;j<NB;j++)
      #pragma xlang name kloop
      for (k=0;k<NB;k++) {
         c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
#pragma xlang transform stripmine iloop NU NUloop
#pragma xlang transform stripmine jloop MU MUloop
#pragma xlang transform interchange kloop MUloop
#pragma xlang transform interchange jloop NUloop
#pragma xlang transform interchange kloop NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform fullunroll NUloop
#pragma xlang transform scalarize_in b in kloop
#pragma xlang transform scalarize_in a in kloop
#pragma xlang transform scalarize_in&out c in kloop
#pragma xlang transform lift kloop.loads before kloop
#pragma xlang transform lift kloop.stores after kloop
```

NU = 1
MU = 4

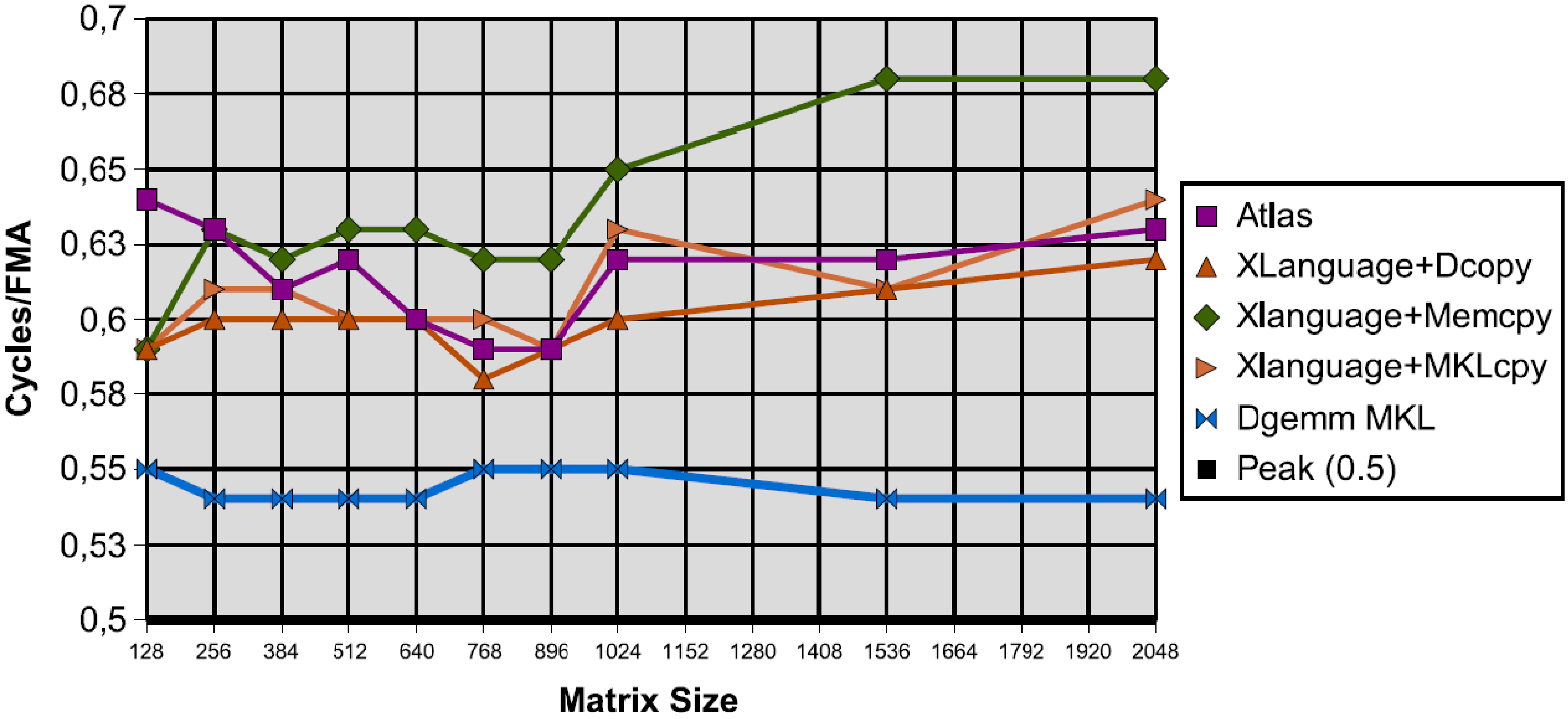
Tiling iloop and jloop

Full Unroll the new tiles

Control the Loads and Stores

Experimental Results(ctd)

2x Intel Itanium 2(Madison) 1.3Ghz, 256KB L2 and 1.5MB L3



Conclusion

- **Pro**
  - Easy to Generate Multiple Program Versions
  - No Knowledge of Compiler Internals necessary
  - Precise Specification of Transformations
  - Defining of new Transformations
  - Macros and Pragmas
- **Contra**
  - No Dependence Analysis
  - No Type Safety
  - Clear Focus on Loops
  - Programmer has to do the Job
  - Gets difficult to read and understand
  - Error prone
  - Exhaustive Search

Questions?