

Iterative Compilation with Kernel Exploration

Denis Barthou¹ Sébastien Donadio^{1,2} Alexandre Duchateau¹
William Jalby¹ Eric Courtois³

¹Université de Versailles, France

²Bull SA Company, France

³CAPS Entreprise, France

LCPC 2006



Kernel decomposition

- Position of the problem
- Kernel decomposition
- Experimental results on kernels
- Combining kernels
- Performance prediction and experimental results
- Concluding remarks



Position of the Problem

High performance linear algebra library for monocoore architecture

- Automatic generation: ATLAS, PhiPAC
 - ▶ Algorithmic knowledge
 - ▶ Optimizes cache usage, empirical search or model driven
- Hand-tuned assembly: constructor library (MKL, ESSL), Goto's BLAS



Position of the Problem

High performance linear algebra library for monocoore architecture

- Automatic generation: ATLAS, PhiPAC
 - ▶ Algorithmic knowledge
 - ▶ Optimizes cache usage, empirical search or model driven
- Hand-tuned assembly: constructor library (MKL, ESSL), Goto's BLAS

Yet performance of hand-tuned code better than ATLAS.

Is it possible to bridge the performance gap without assembly code?



Principle of Decomposition

Feed the compiler with code easy to optimize.

1 Tile for memory reuse, if needed

- ▶ Use X-language pragmas.

LCPC05: A language for the compact representation of multiple program versions,

S.Donadio, J.Brodman, T.Roeder, K.Yotov, D.Barthou, A.Cohen, M.-J.Garzarán, D.Padua and K.Pingali



Principle of Decomposition

Feed the compiler with code easy to optimize.

- 1 Tile for memory reuse, if needed
 - ▶ Use X-language pragmas.
LCPC05: A language for the compact representation of multiple program versions,
S.Donadio, J.Brodman, T.Roeder, K.Yotov, D.Barthou, A.Cohen, M.-J.Garzarán, D.Padua and K.Pingali
- 2 Decompose computation of the tiles into simple kernels
 - ▶ kernels tile the computation,
 - ▶ kernels are independent from application context,
 - ▶ experimental search for the best kernel, in vitro.
- 3 Combine best kernel(s) based on performance prediction



Kernel Decomposition

Kernel features

- Inner-most loops of the computation
→ tiling 3D loops with 1D, 2D or 3D tiles.
- Constant loop bounds
- Arrays simplified to kernel working set



Kernel Decomposition

Kernel features

- Inner-most loops of the computation
→ tiling 3D loops with 1D, 2D or 3D tiles.
- Constant loop bounds
- Arrays simplified to kernel working set

Detailed decomposition

- Perform loop transformations on tile (interchange, unroll, ...)
- Select inner most loops
- Simplify arrays and specialize loop bounds



Kernel Decomposition

Kernel features

- Inner-most loops of the computation
→ tiling 3D loops with 1D, 2D or 3D tiles.
- Constant loop bounds
- Arrays simplified to kernel working set

Detailed decomposition

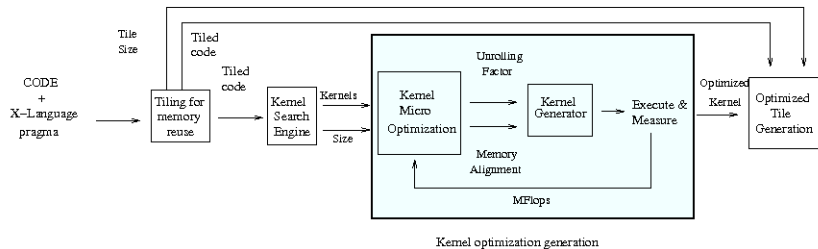
- Perform loop transformations on tile (interchange, unroll, ...)
- Select inner most loops
- Simplify arrays and specialize loop bounds

Kernel parameters

- Unrolling factors
Impacts IPC
- Array alignment
Impacts memory banks accessed
- Loop bounds
Impacts optimizations selected by compiler, memory banks accessed



Kernel Decomposition



Matrix Multiplication Kernels

Original tile

```
for (ii = 0; ii < NI; ii++)  
  for (jj = 0; jj < NJ; jj++)  
    for (kk = 0; kk < NK; kk++ )  
      c[ii][jj] += a[ii][kk] * b[kk][jj];
```



Matrix Multiplication Kernels

Original tile

```
for (ii = 0; ii < NI; ii++)  
  for (jj = 0; jj < NJ; jj++)  
    for (kk = 0; kk < NK; kk++ )  
      c[ii][jj] += a[ii][kk] * b[kk][jj];
```

After unroll of kk loop and interchange jj,kk

```
for (ii = 0; ii < NI; ii++)  
  for (kk = 0; kk < NK; kk+=2 )  
    for (jj = 0; jj < NJ; jj++)  
      c[ii][jj] += a[ii][kk] * b[kk][jj];  
      c[ii][jj] += a[ii][kk+1] * b[kk+1][jj];
```



Matrix Multiplication Kernels

Original tile

```
for (ii = 0; ii < NI; ii++)
  for (jj = 0; jj < NJ; jj++)
    for (kk = 0; kk < NK; kk++ )
      c[ii][jj] += a[ii][kk] * b[kk][jj];
```

After unroll of kk loop and interchange jj,kk

```
for (ii = 0; ii < NI; ii++)
  for (kk = 0; kk < NK; kk+=2 )
    for (jj = 0; jj < NJ; jj++)
      c[ii][jj] += a[ii][kk] * b[kk][jj];
      c[ii][jj] += a[ii][kk+1] * b[kk+1][jj];
```

Extracted kernel: daxpy 2

```
for (jj = 0; jj < 100; jj++)
  c[jj] += a0 * b0[jj];
  c[jj] += a1 * b1[jj];
```



Matrix Multiplication Kernels

All kernels of matrix multiplication are among

- daxpy (1D)
- dot product (1D)
- matrix-vector product (2D)
- outer product (2D)
- matrix-matrix product (3D)



Matrix Multiplication Kernels

Decomposition implementation

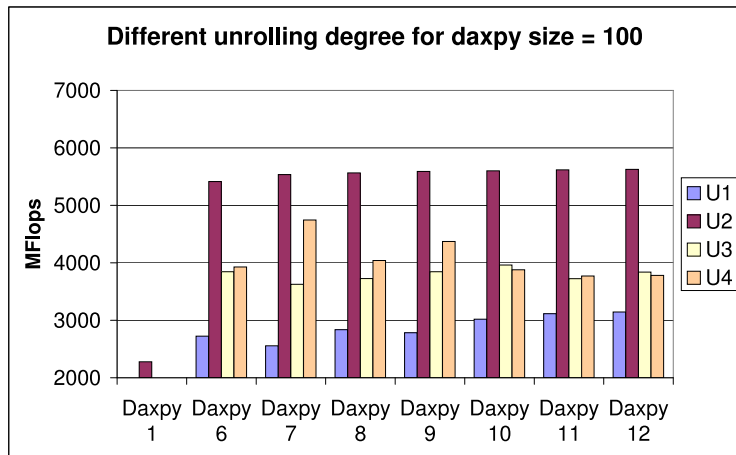
- Implemented as a directive of X-language
`#pragma xlang decompose(# of loops).`

Experimental Setup

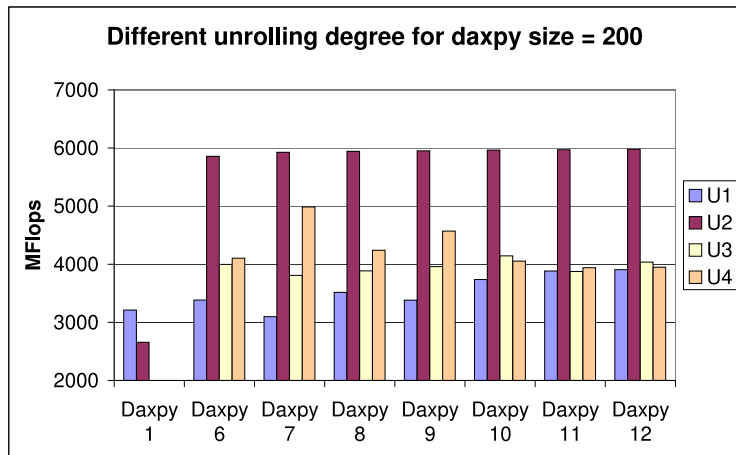
- Novascale Bull server, 1.6Ghz Itanium 2 processors,
- 256KB level 2 cache, 9MB level 3 cache.
- Intel C compiler 9.0 with 03 `fnoaliases`



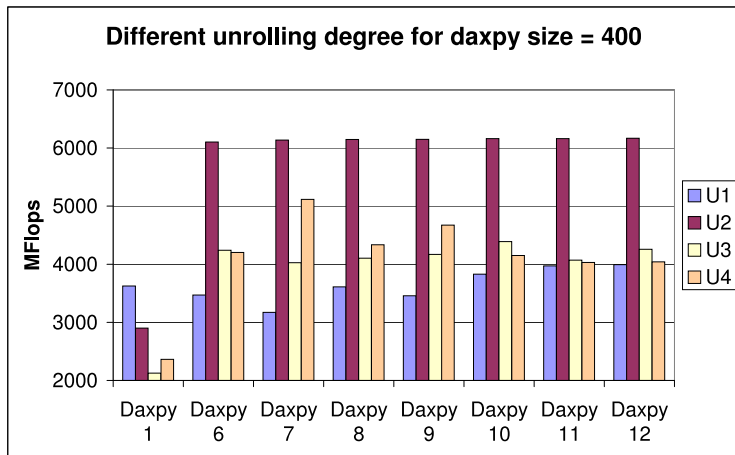
Daxpy Performance



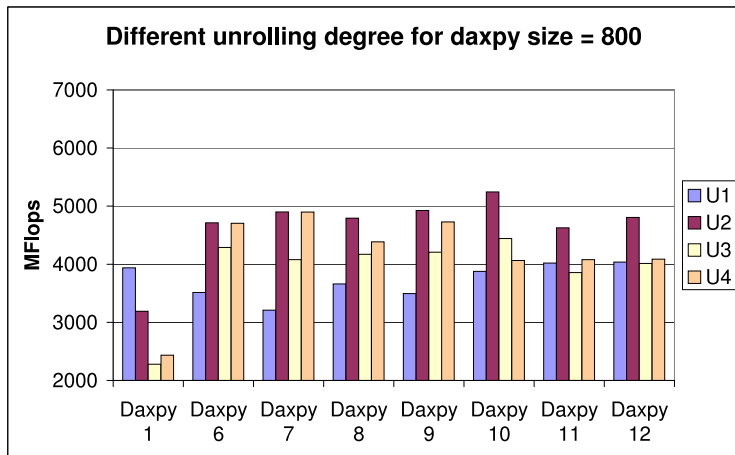
Daxpy Performance



Daxpy Performance



Daxpy Performance



Performance Prediction

Performance can be predicted

If kernels have same in vivo/in vitro same execution context:

- Data in cache
- Same data layout
- Same alignment conditions



Performance Prediction

Performance can be predicted

If kernels have same in vivo/in vitro same execution context:

- Data in cache
- Same data layout
- Same alignment conditions

Benchmarked memory copies

- Resize arrays (copies or transpositions)
- Align for best performance
- If not used (no reuse, no resize), worst alignment considered



Putting Kernels to Work

Tiling with kernels

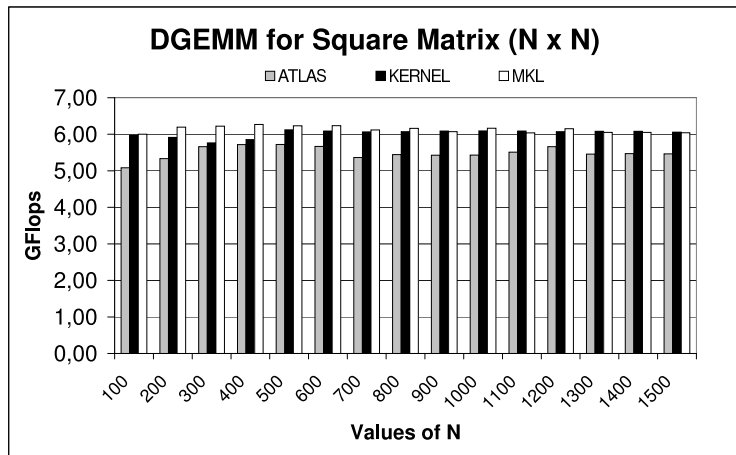
- Kernel size must fit into tile
 - ▶ Different kernel decompositions according to matrix size
- All copies hoisted at beginning of tile

Performance model

- Simple addition of kernel and copies performance

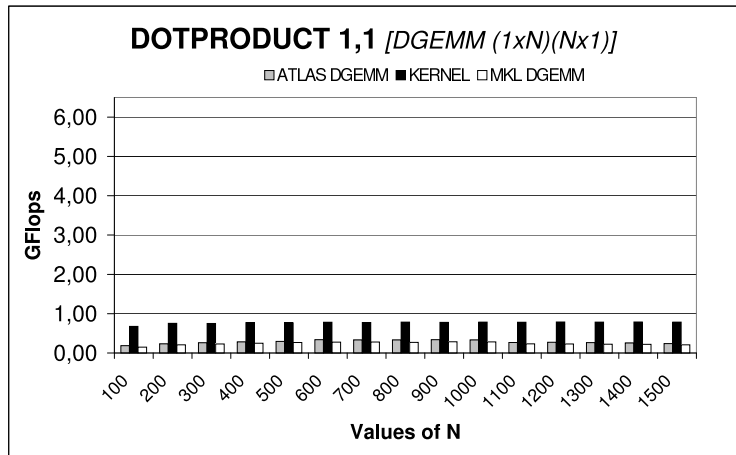


Experimental Results on DGEMM



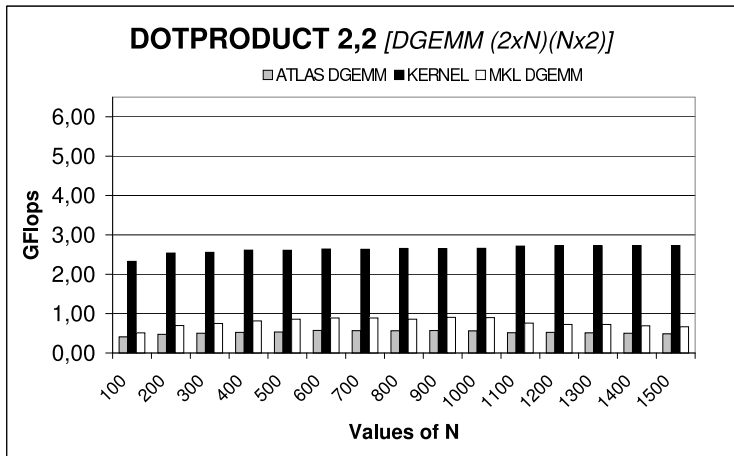
Experimental Results on DGEMM for thin matrices

(only tile performance measured)



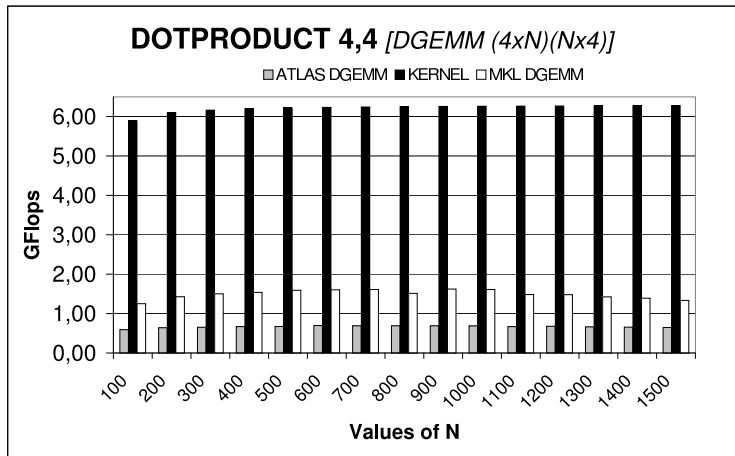
Experimental Results on DGEMM for thin matrices

(only tile performance measured)



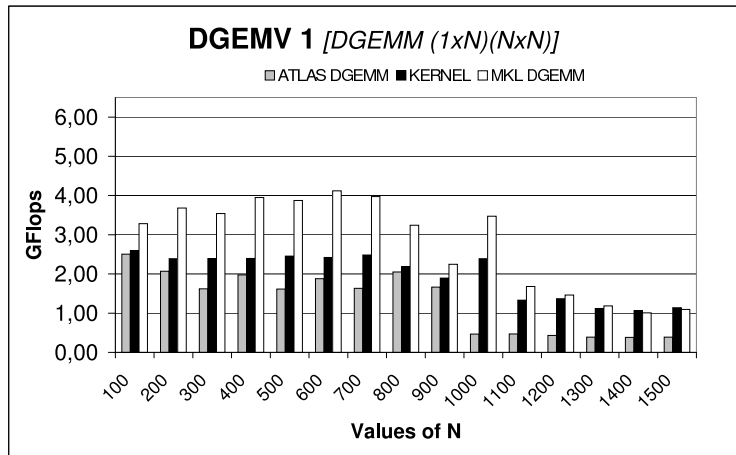
Experimental Results on DGEMM for thin matrices

(only tile performance measured)



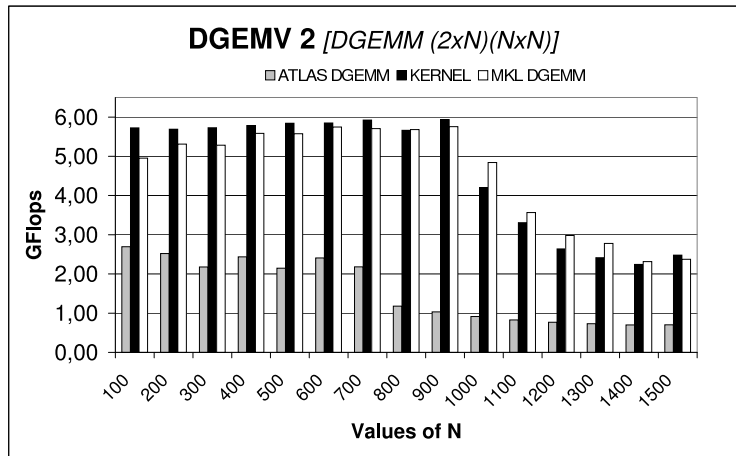
Experimental Results on DGEMM for thin matrices

(only tile performance measured)



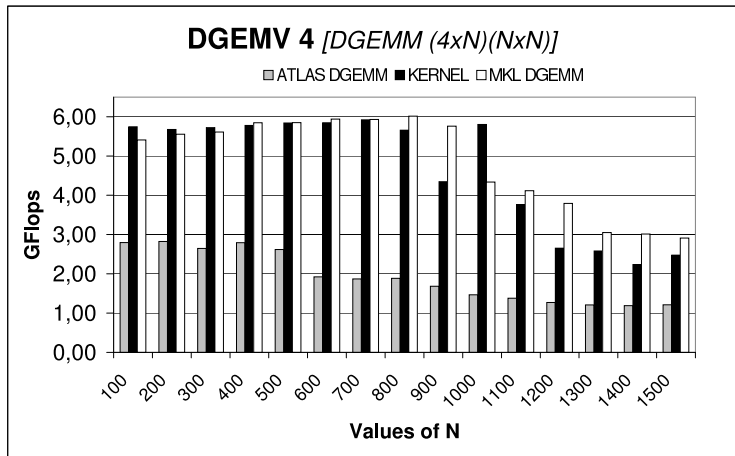
Experimental Results on DGEMM for thin matrices

(only tile performance measured)



Experimental Results on DGEMM for thin matrices

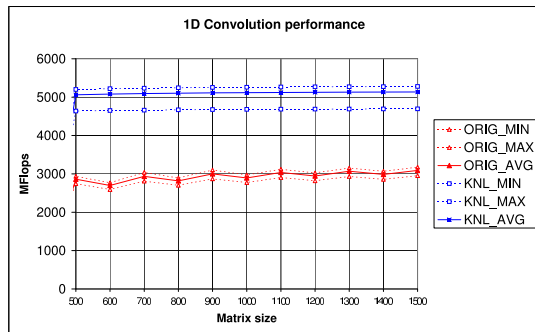
(only tile performance measured)



Kernel Decomposition applied on 1D Convolution

1D Convolution

```
for(i=0;i<N-n;i++)  
  for(j=0;j<2*n;j++)  
    a[i] += b[j] * c[i-j+n];
```



Conclusion/Future works

- No assembly code
 - ▶ But compiler dependent
- Very competitive with MKL, outperforming ATLAS
- Approach not application dependent
 - ▶ Need to be applied on other library codes
- Apply to other architectures
- Issue of finding appropriate higher level optimization still remains.



Thank you !

