Iterative Compilation with Kernel Exploration

Denis Barthou
denis.barthou@prism.uvsq.fr

joint work with

Sébastien Donadio, Alexandre Duchateau, William Jalby
Université de Versailles

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Context

High Performance Computing

- Larger and larger field of application

  simulation (biology, aeronautics, drug design,...), multimedia applications, large data computations

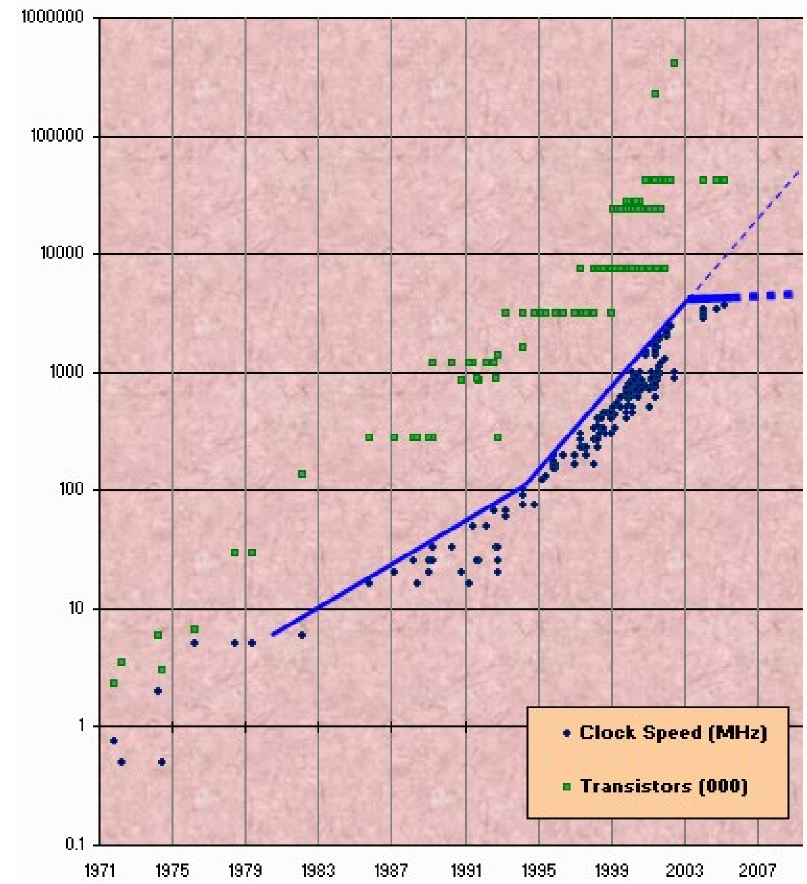
- Higher level of parallelism
  - Numa architectures
  - Hardware accelerators (GPU, clearspeed,...)
  - Multicore
  - Large IPC

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Context

Evolution of Moore's Law:

- end of performance for free ?
- more effort to tap future computational resources
- more need for optimizations



Issues in compilation

Major compiler problems:

- Dealing with more and more complex hardware (ILP, speculation, out of order, multilevel caches)
- → Specific, complex optimizations
- Optimizations not always beneficial for performance (fusion)
- Complex interactions between optimizations
- → Heuristic driven optimization sequences

=> As a result, performance of compiler generated code even for simple matrix multiplication is disappointing

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

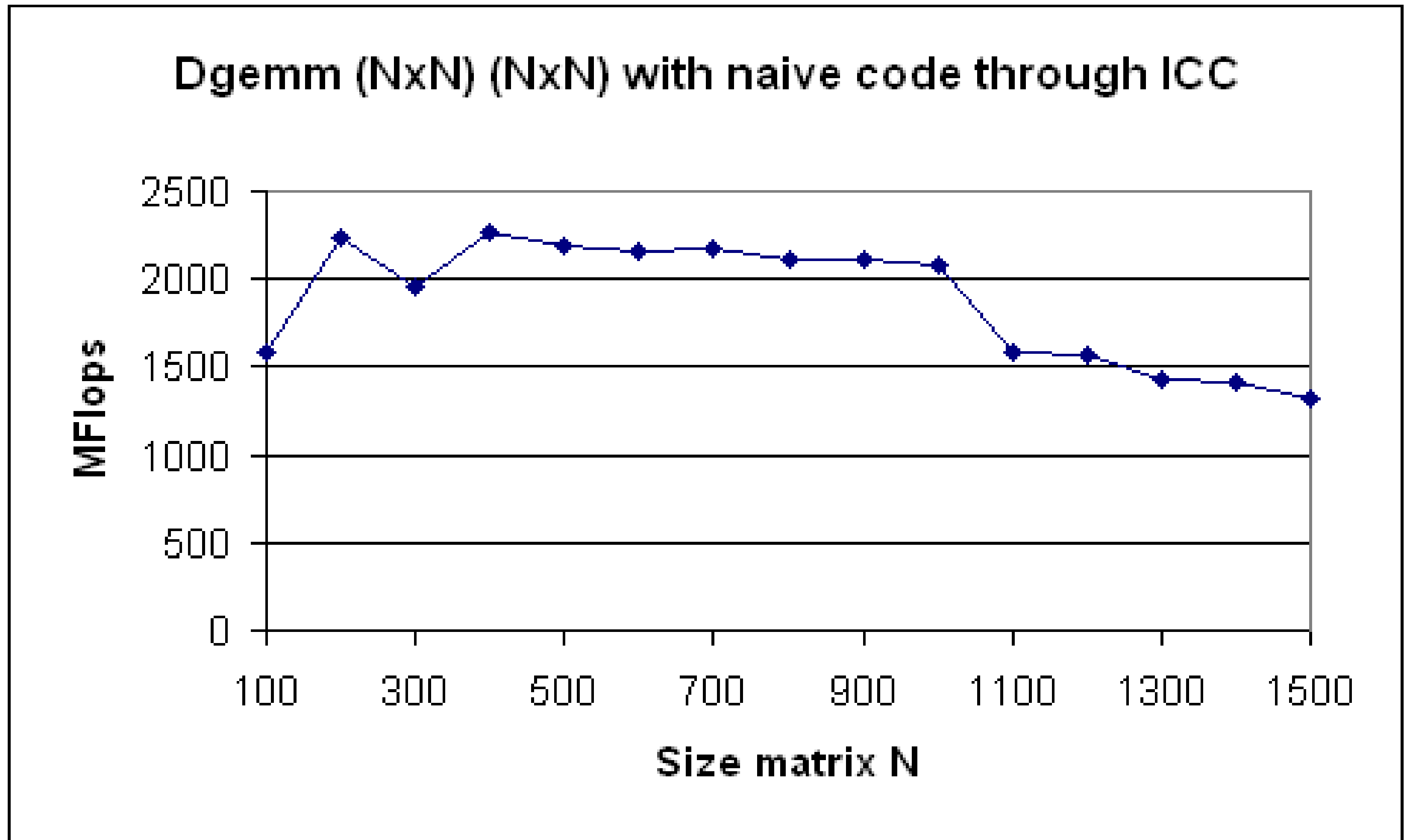
Experimental setup

- ICC v9.0 (build 20050430)
  - \- O3 – no alias

- Itanium 2 Bull NOVASCALE
  - 1.575 GHz
  - 9MB L3
  - Peak performance = 6300 Mflops (1 FMA every 0.5 cycle)

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Issues in compilation



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outline

- Exploring optimization space
  - **Manually: Multistage compilation languages**
  - Automatically: by search or with a model
  - Combine: Kernel decomposition
- Feed-back guided code tuning

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Multistage compilation languages

Goal : **Adapt optimization sequence to application**

Apply transformations to specific code fragments

- pragmas (icc black belts, X language)
- languages manipulating code objects (meta ocaml, `C, Taskgraph, ccg)

Issues

- correction of generated code, debug
- ease of use

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language

Features:

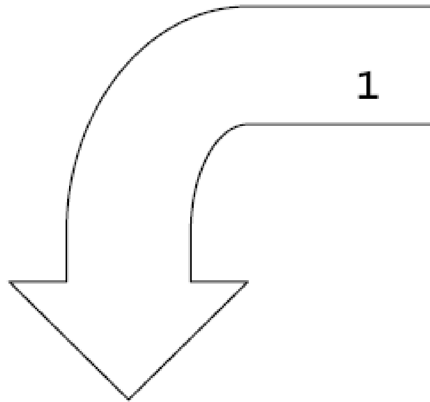
- Language of pragmas
- Use elementary transformations, with their parameter

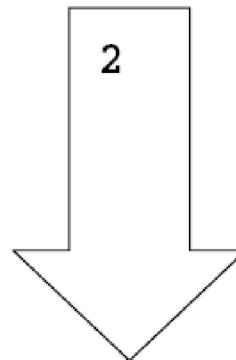
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

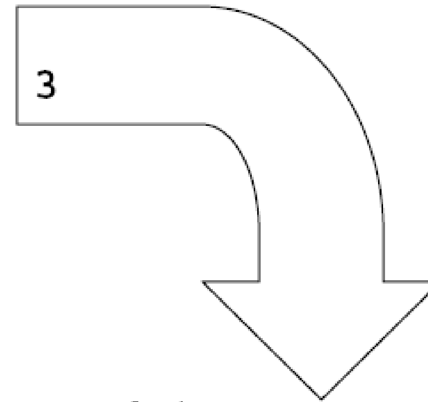
X language

```
do i=1, 100
     a(i)=b(i)+c(i)
end do
```

Unroll








```
do i=1, 100
     a(i)=b(i)+c(i)
end do
```

```
do i=1, 100, 2
     a(i)=b(i)+c(i)
     a(i+1)=b(i+1)+c(i+1)
end do
```

```
do i=1, 99, 3
     a(i)=b(i)+c(i)
     a(i+1)=b(i+1)+c(i+1)
     a(i+2)=b(i+2)+c(i+2)
end do
a(100)=b(100)+c(100)
```

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language example

```
#pragma xlang parameter U [1:3]
#pragma xlang name loop1
for (i=0; i<100; i++)
  a(i) = b(i) + c(i)
#pragma xlang transform unroll loop1 U
```

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language

Features:

- Language of pragmas
- Use elementary transformations, with their parameter
- Compose elementary transformations

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language

```
for (i=0; i<n*2; i++)
  for (j=0; j<m; m++)
    a(i) = a(i) + b(j)
```

outer unroll

```
for (i=0; i<n*2; i++)
  for (j=0; j<m; j++)
    a(i) = a(i) + b(j)
  for (j=0; j<m; j++)
    a(i+1) = a(i+1) + b(j)
```

fusion

```
for (i=0; i<n*2; i++)
  for (j=0; j<m; j++)
    a(i) = a(i) + b(j)
    a(i+1) = a(i+1) + b(j)
```

stripmine

```
for (i=0; i<n*2; i+=2)
  for (ii=i; ii<i+2; ii++)
    for (j=0; j<m; j++)
      a(ii) = a(ii) + b(j)
```

interchange

```
for (i=0; i<n*2; i+=2)
  for (j=0; j<m; j++)
    for (ii=i; ii<i+2; ii++)
      a(ii) = a(ii) + b(j)
```

inner unroll

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language example

```
#pragma xlang name loopi
for (i=0; i<n*2; i++)
#pragma xlang name loopj
  for (j=0; j<m; j++)
    a(i) = a(i) + b(j)
#pragma xlang transform stripmine loopi 2 loopii
#pragma xlang transform interchange loopii loopj
#pragma xlang transform unroll loopii 2
```

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language

Features:

- Language of pragmas
- Use elementary transformations, with their parameter
- Compose elementary transformations
- User-defined transformations

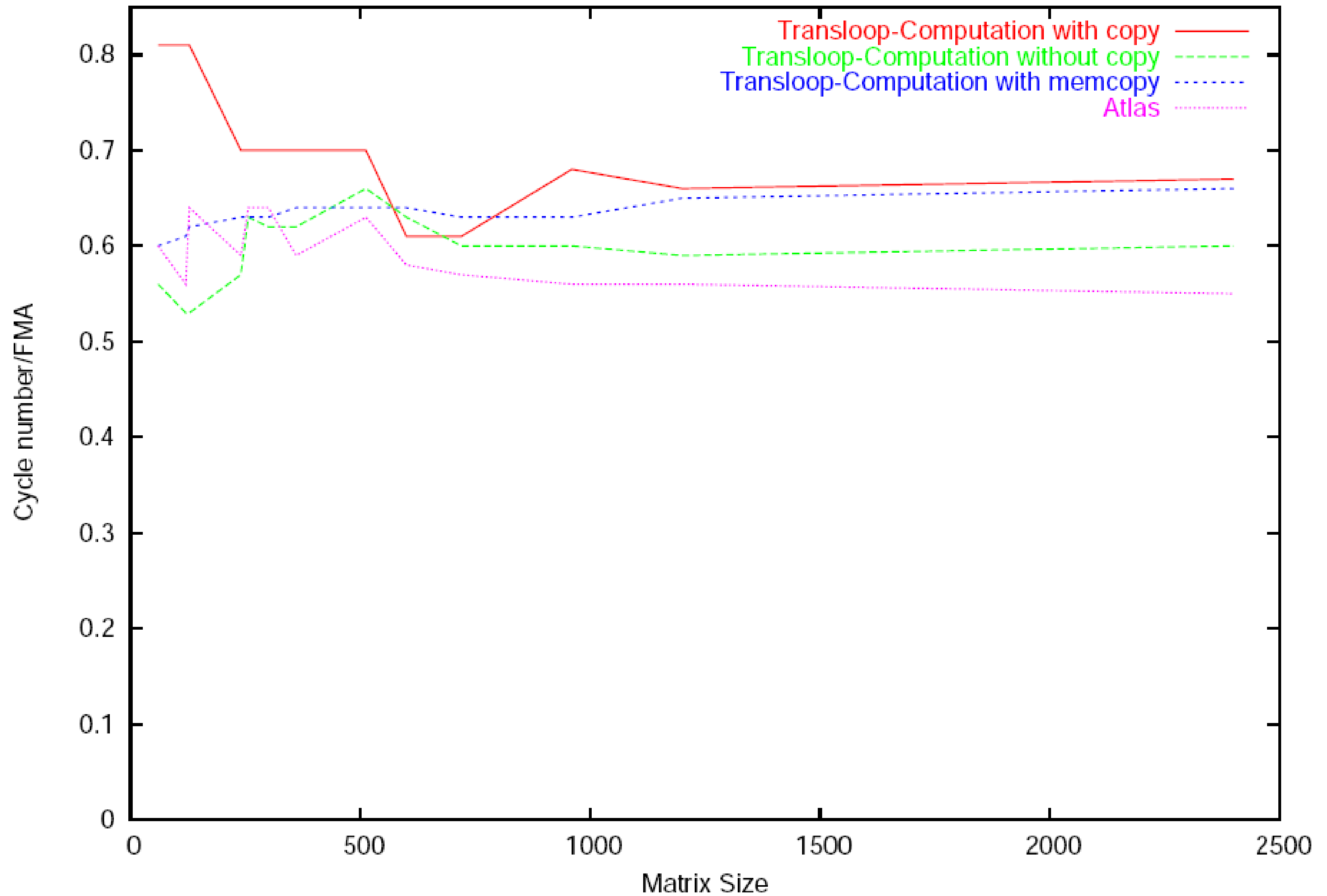
X language example

Definition of fusion by pattern matching rule (Prolog)

```
optimize(fusion(I,J),
           for(I,LB,UB,S,B); for (J,LB,UB,S,B2),
          for(I,LB,UB,S,B;B3)):-
       substitute(I/J,B2,B3).
```

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language applied to dgemm



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language

Pros:

- Adapt optimization to the application
- New optimizations easily added, pattern matching rules
- Description of the iteration space
- Not application specific, possible to obtain ATLAS performance (not unleashed)

Good language to develop compilation strategies

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

X language

Cons:

- Programmer does the job of the compiler! (general to multistage compilation languages)
- Not type safe
- No validation of transformations
- Difficult to read after few lines

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outline

- Exploring optimization space
  - Manually: Multistage compilation languages
  - **Automatically: by search or with a model**
  - Combine: Kernel decomposition
- Feed-back guided code tuning

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Iterative compilation

Cf previous courses.

- Space of optimizations:
  - optimization sequences
  - parameters (unroll factor, tile sizes, prefetch distance,...)
- Search guided
  - by execution results (Atlas, FFTW, Spiral)
  - by model (Atlas with model)
  - by knowledge of application semantics (all of these)

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Iterative compilation

Pros:

- Source to source transformations (portable)
- May use semantics of the application

Cons:

- Source to source transformation (compiler dependent)
- **Still outperformed by 20% by handtuned codes**

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outline

- Exploring optimization space
  - Manually: Multistage compilation languages
  - Automatically: by searching or with a model
  - **Combine: Kernel decomposition**
- Feed-back guided code tuning

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Kernel decomposition

Objectives:

- Source to source transformations only
- Focus on linear algebra codes
- Close the gap with hand tuned codes

Idea:

Compilers are our friends, use them to their best

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Target typical Code: Matrix Multiplication

Interesting features:

- Contains all of the typical ingredients: ILP, locality, generic shapes,
- Large amount of research work available

Competitors:

- MKL: highly hand tuned library developed by INTEL
- ATLAS: automatically generated library
- GOTO libraries: very high performance assembly coded library
- Close to optimal performance obtained by GOTO and MKL

➔ tough challenge, no performance losses allowed

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

General approach

Main steps:

- **High level transformations**, data reuse tiling (if needed)
- **Decompose** data reuse tiles into simple kernels
- → Several decompositions can be obtained
- **Benchmark** all kernels, out of application context
- **Combine** best kernels to generate the code

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Kernel definition

Features of a kernel:

- static control program, constant loop bounds
- bounded loop nest depth
- all working set is in cache
- compact array structures to the working set used in the kernel

→ kernels are simple codes for the compiler to optimize

→ makes kernels executable out of application context

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Kernel decomposition

Search transformations:

- loop interchange, fusion/fission, unroll&jam
- for now, exhaustive search...

Select sub loop nest

- starting from 3 loops, generates kernels of 1,2 and 3 loops

Simplify array structures

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Decomposition applied to dgemm

Starting with a mini mmm tile.

External tiling obtained with X language.

```
for i = 0, N1
    for i = 0, N2
     for k = 0, N3
          C[i][j] = C[i][j] + A[i][k] * B[k][j]
```

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Decomposition applied

1st example: no transformation, select inner loop

```
for i = 0, N1
  for j = 0, N2
     for k = 0, N3
        C[i][j] = C[i][j] + A[i][k] * B[k][j]
```

Simplifies into:

```
for k = 0, N3
  C = C + A[k] * B[k]
```

➔ This is a dot product

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Decomposition applied

2nd example: interchange i,k, unroll k by 2, select inner loop

```
for i = 0, N1
    for k = 0, N3,2
        for j = 0, N2
            C[i][j] = C[i][j] + A[i][k] * B[k][j]
            C[i][j] = C[i][j] + A[i][k+1] * B[k+1][j]
```

Simplifies into:

```
for j = 0, N2
    C[j] = C[j] + A0 * B0[j]
    C[j] = C[j] + A1 * B1[j]
```

Variant of daxpy

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Decomposition applied

All kernels found by decomposition (variants)

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

Unrolling the kernel itself is also considered.

Implementation:

- decomposition automatic with one elementary transformation of X language
- tiling and other high level transformations with X language

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Benchmarking the kernels

In vitro benchmarking

- In vitro behavior has to be the same than in vivo (inside application)
- All kernel inputs impacting performance are considered.

Search space:

- loop trip counts,
- unrolling factors,
- array alignment

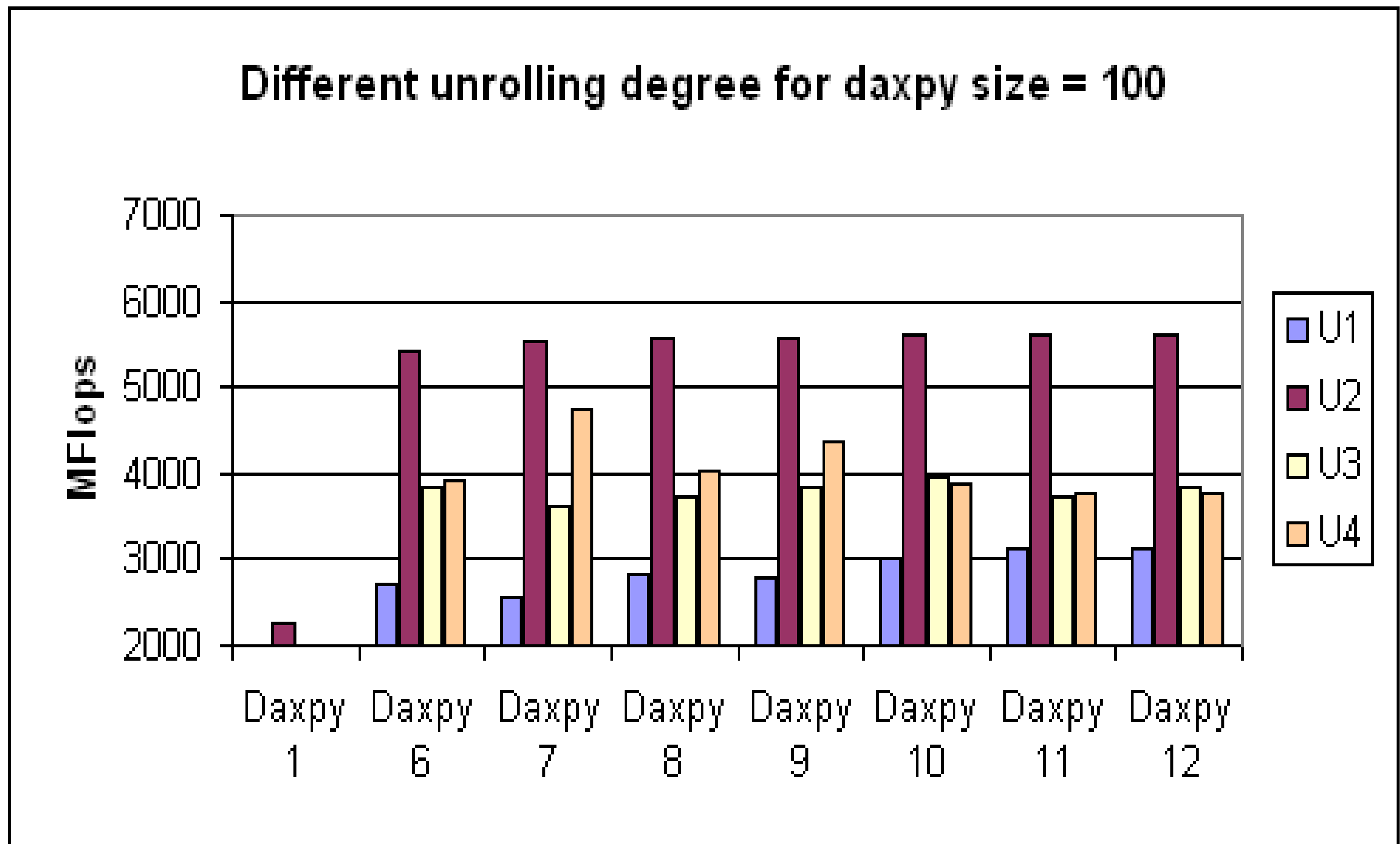
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

DAXPY K kernel

```
for i = 1, 100
    Y[i] = Y[i] + A1*X1[i] + ... + Ak*Xk[i]
```

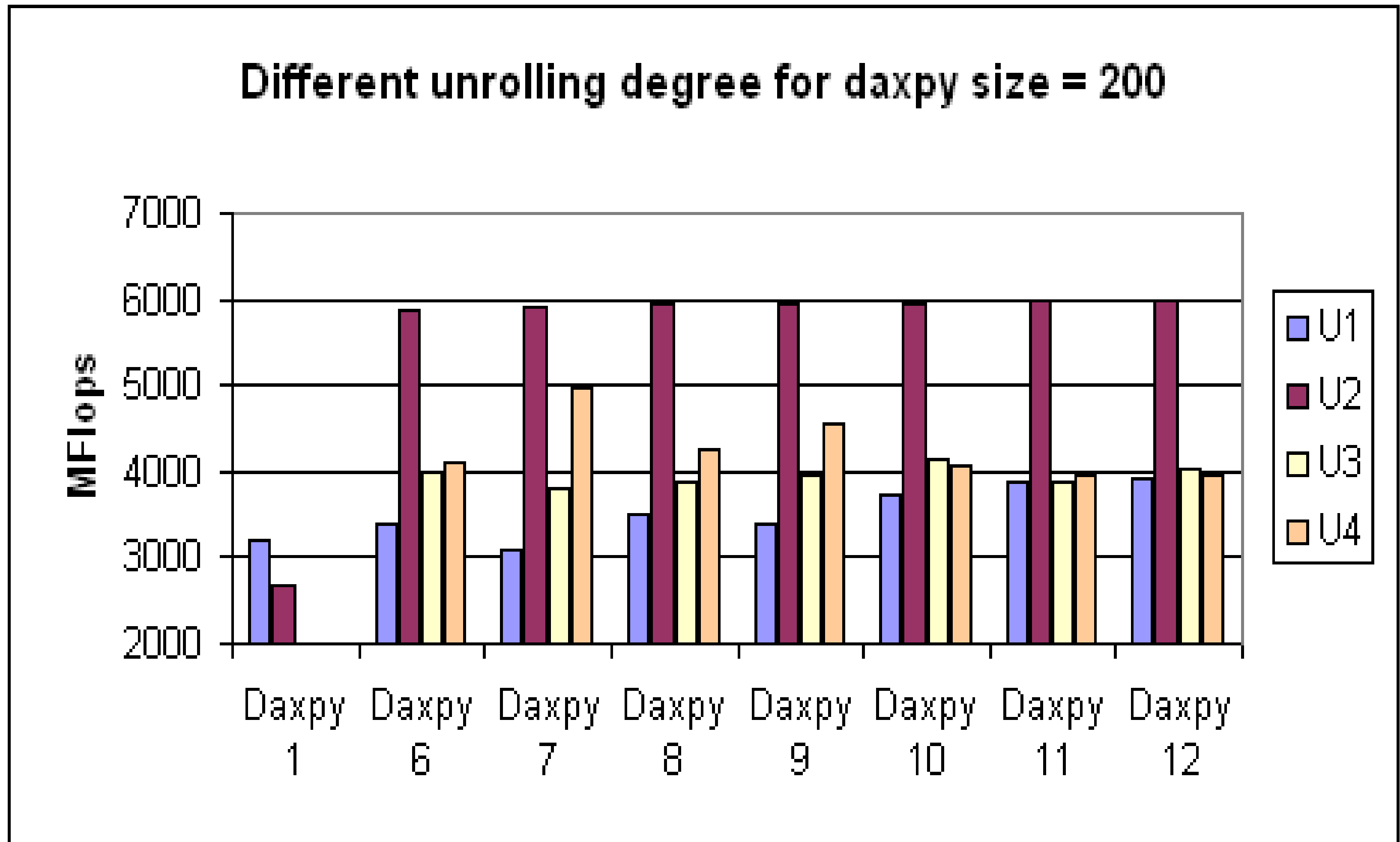
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Daxpy_k Performance (N=100)

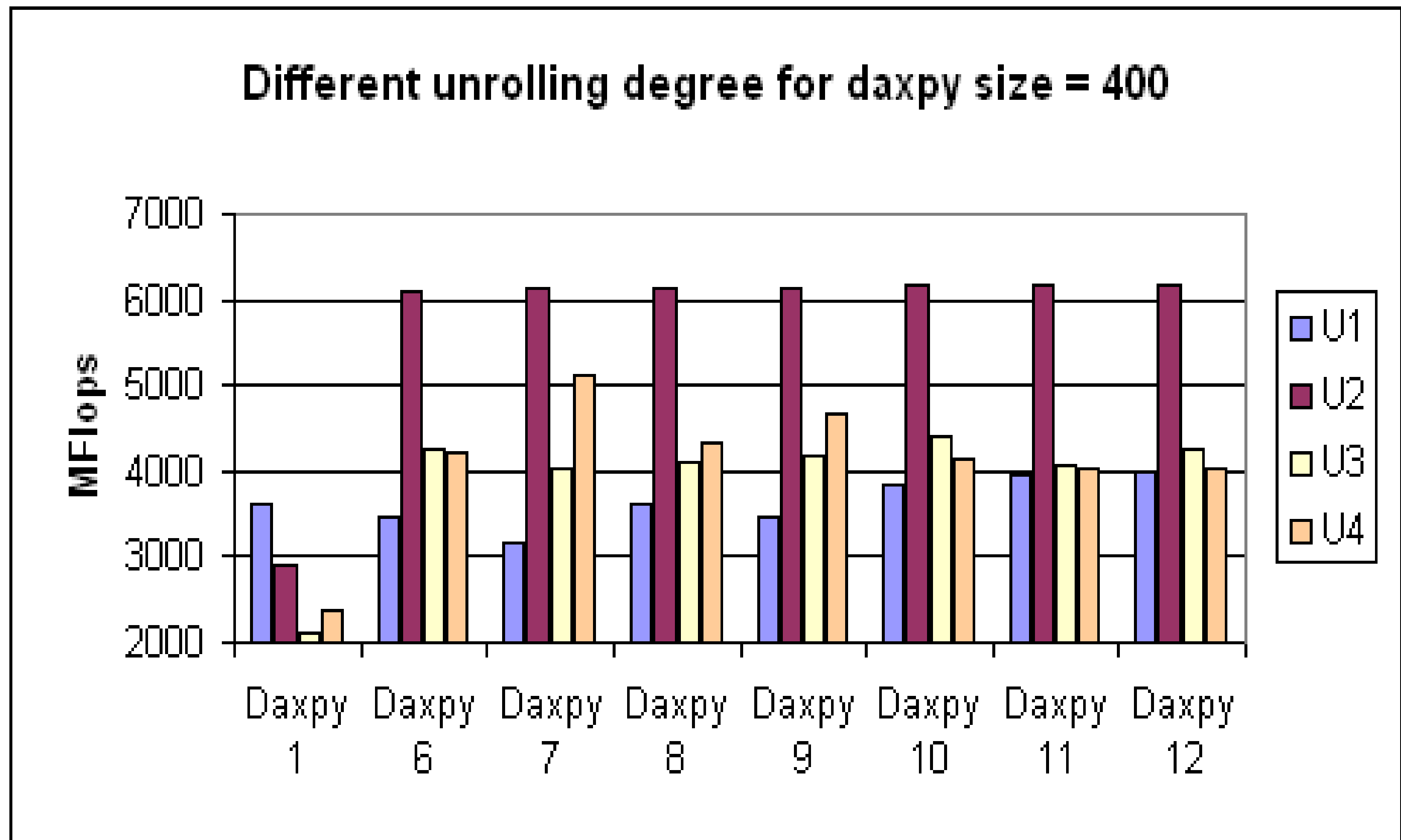


UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

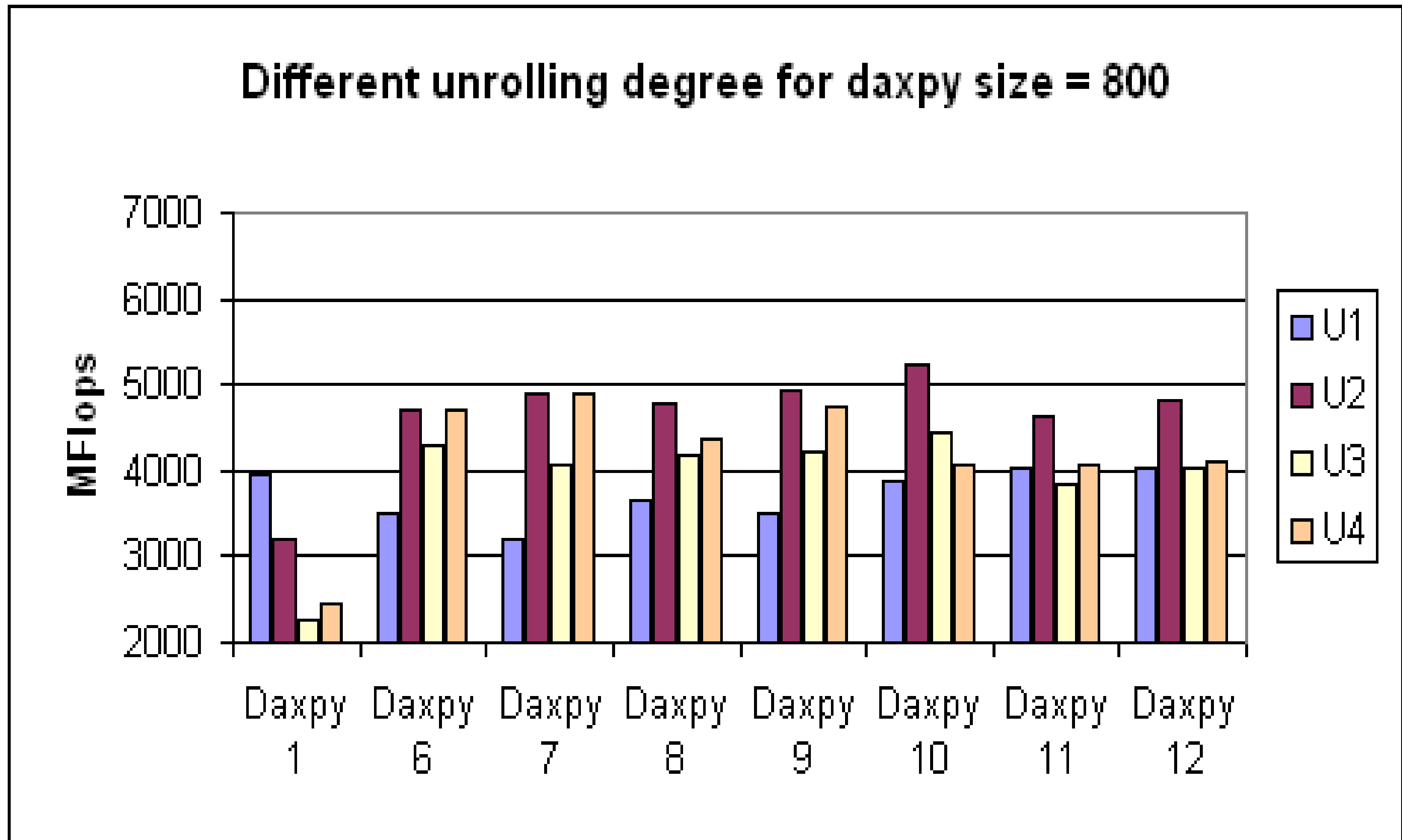
Daxpy_k Performance (N=200)



Daxpy_K Performance (N=400)



Daxpy_k Performance (N=800)



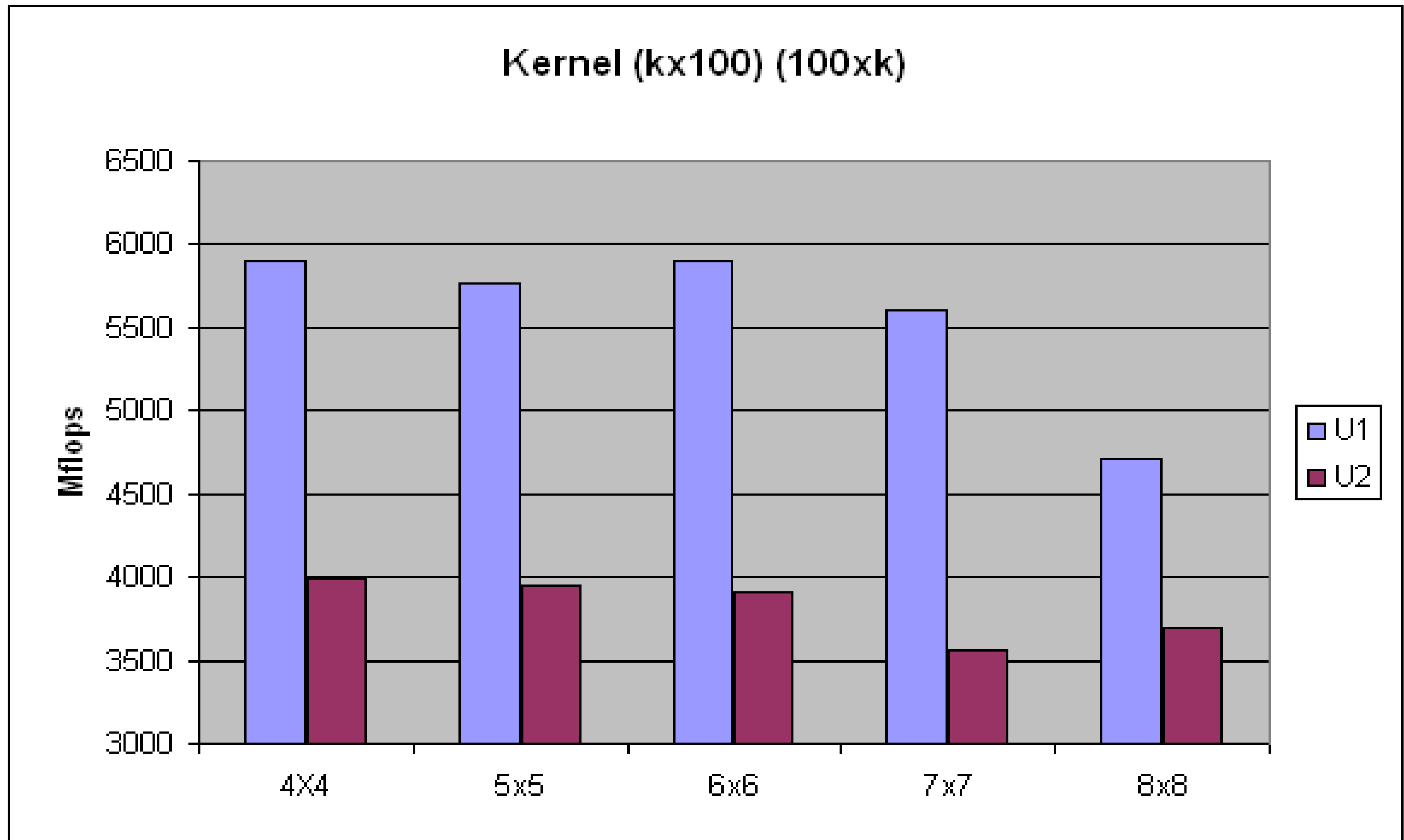
Dot product k kernel

```
For i = 1, 100
   S1 = S1 + X1[i]*Y1[i]
   S2 = S2 + X1[i]*Y2[i]
   S3 = S3 + X2[i]*Y1[i]
   S4 = S4 + X2[i]*Y2[i]
```

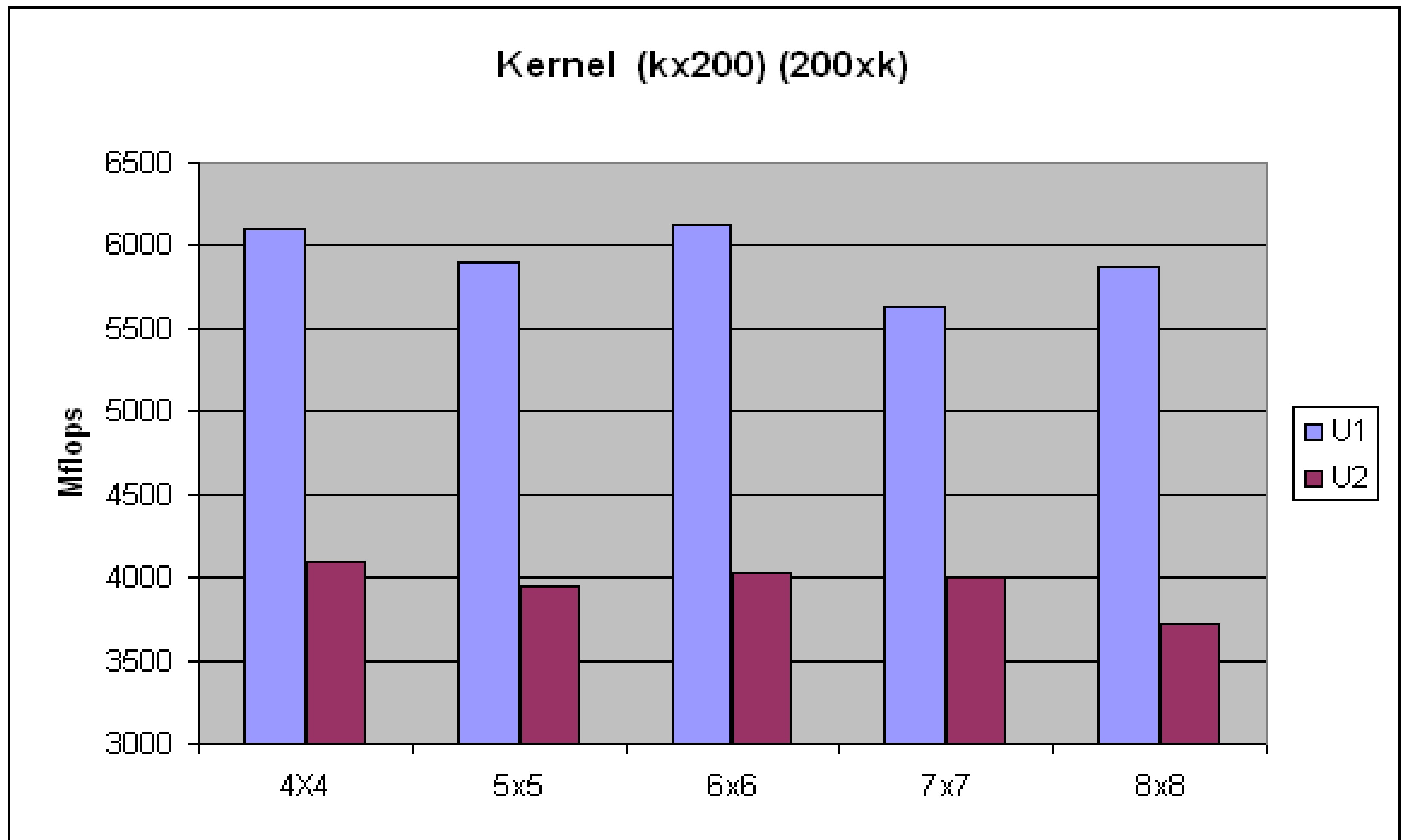
Code above will be called (2xn)(nx2) or simpler 2x2

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Performance of (kxn) (nxk)



Performance of (kxn) (nxk)

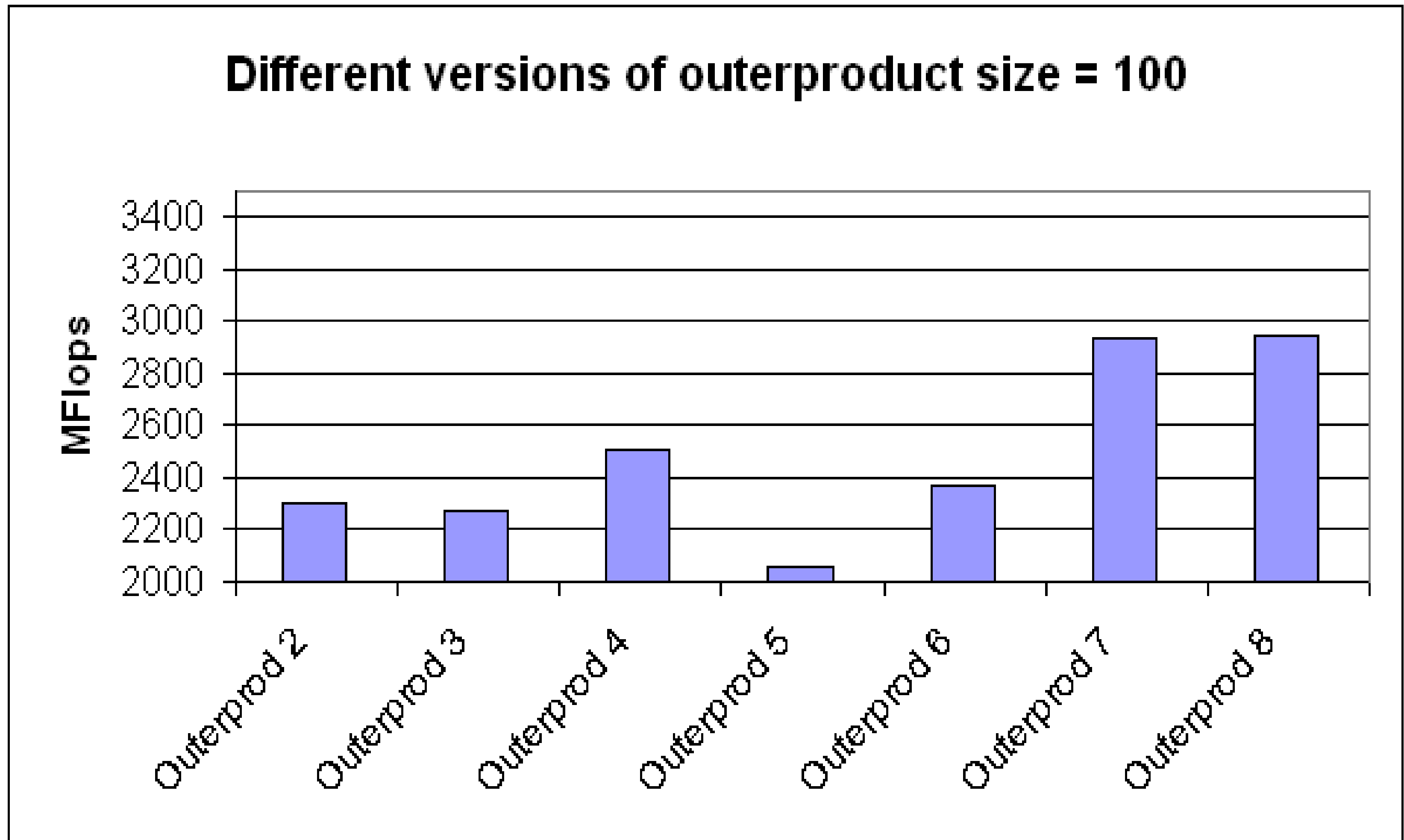


Outer Product K kernel

```
For i = 1, 100
    Y1[i] = Y1[i] + A1*X[i]
    Y2[i] = Y2[i] + A2*X[i]
    .......
    Yk[i] = Yk[i] + Ak*X[i]
```

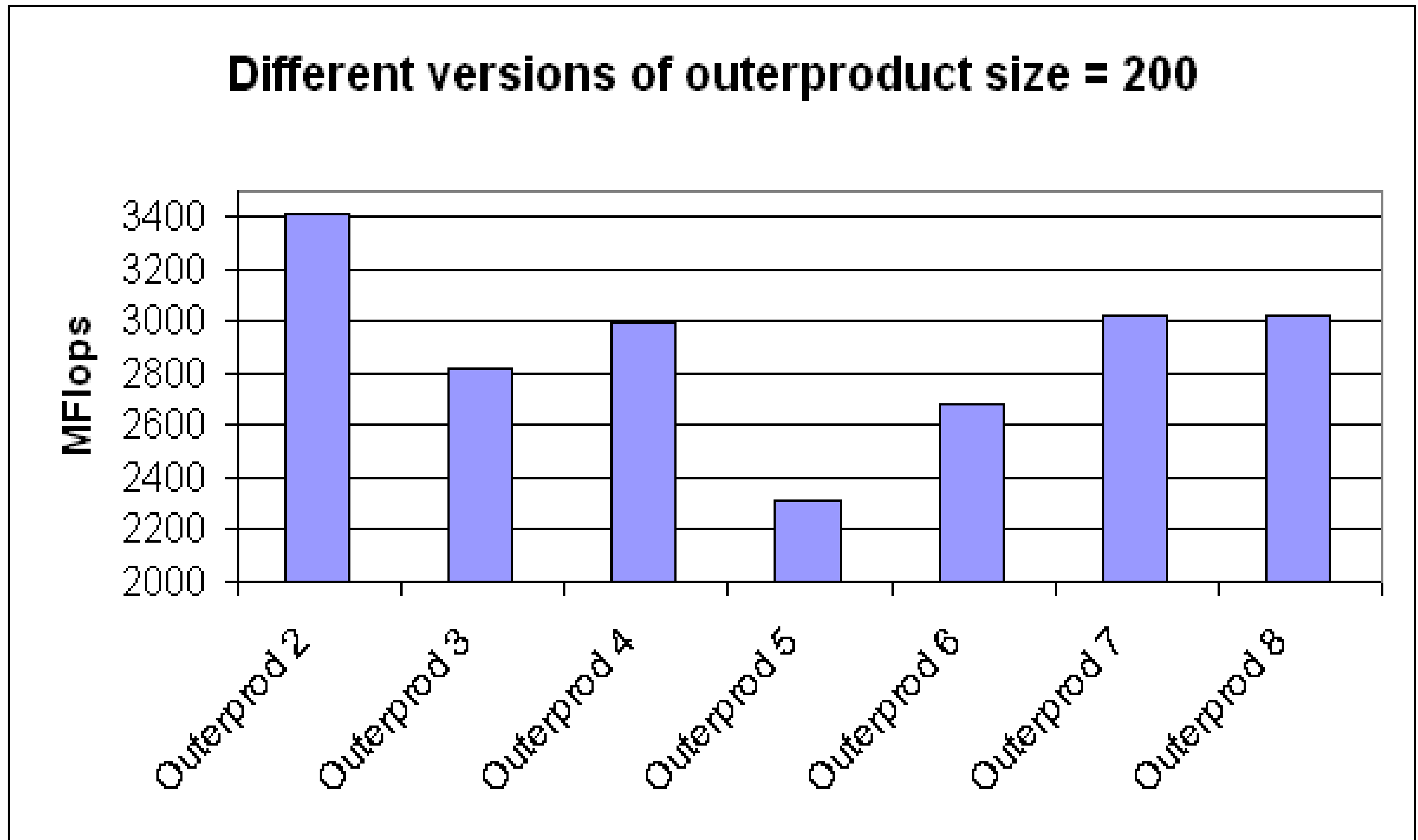
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outer_Product K Performance (N=100)



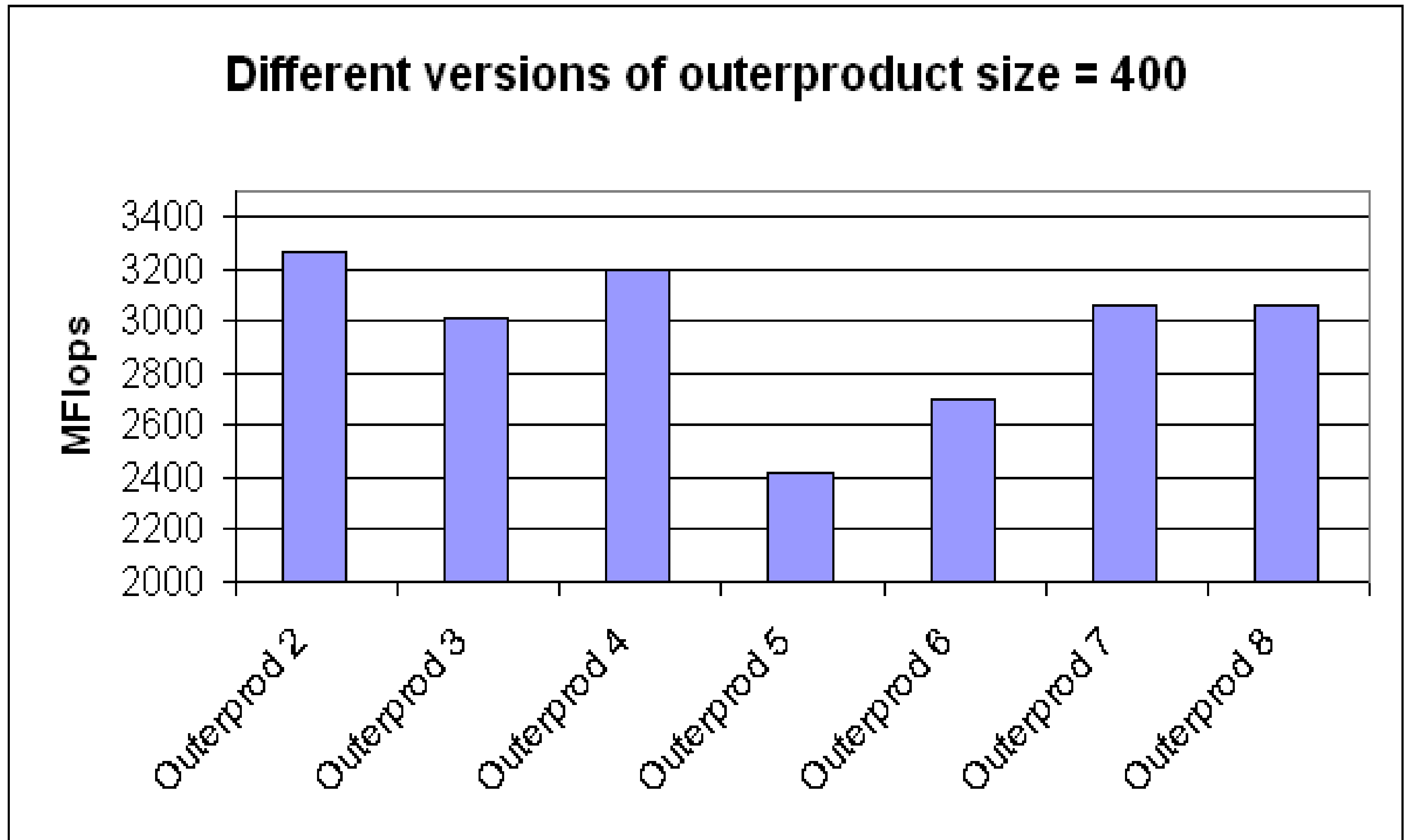
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outer_Product_k Performance (N=200)



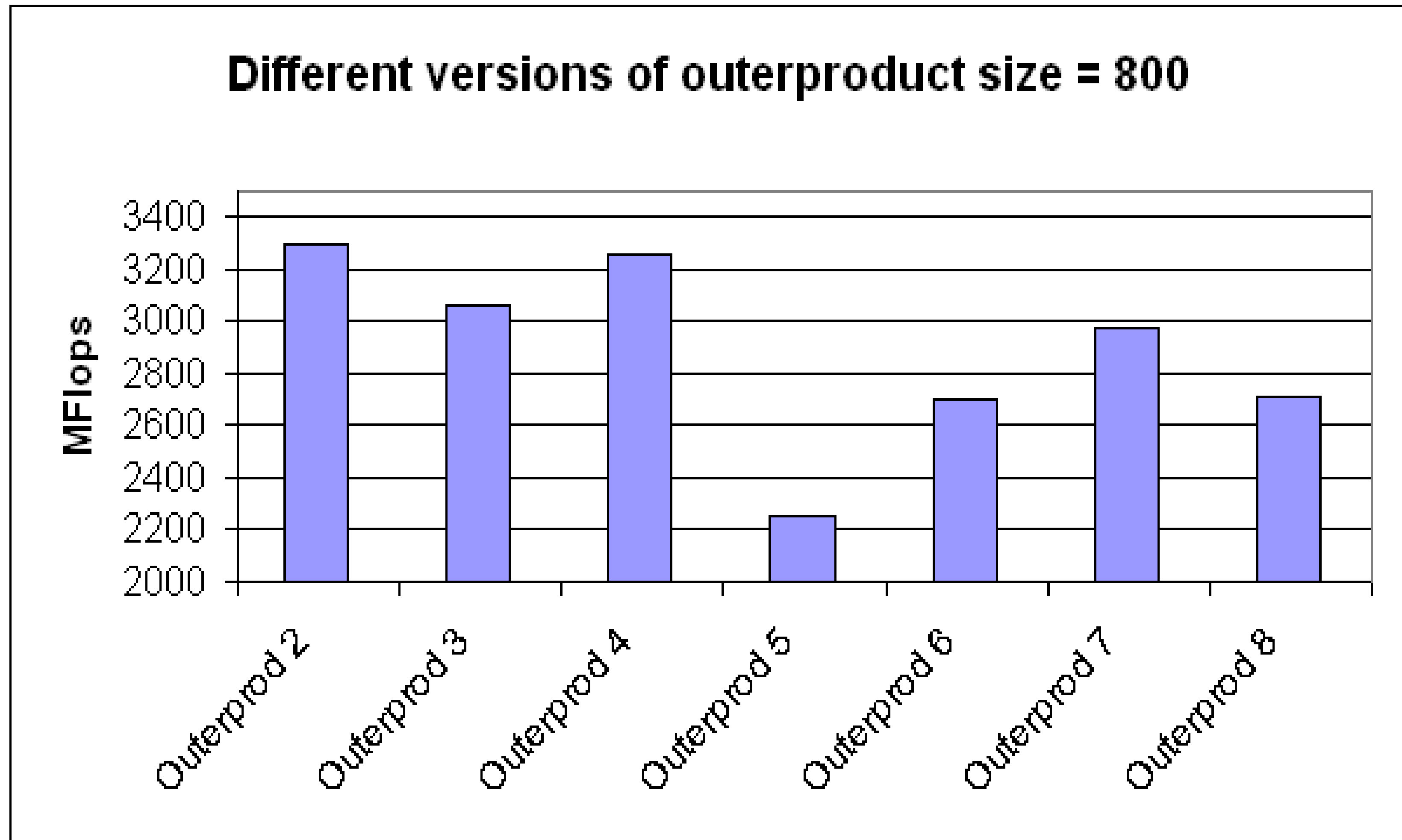
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outer_Product_K Performance (N=400)



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outer_Product_K Performance (N=800)



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Combining kernels

Copying input/output arrays of kernels

- Ensures stability (array alignment)
- Data in cache, same results expected as in vitro.
- Memcopies are benchmarked (worst case measures)

According tile size:

For all kernels and their appropriate memcopies,

- Find out the best combination according to performance model (trivial)

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Combining kernels for dgemm

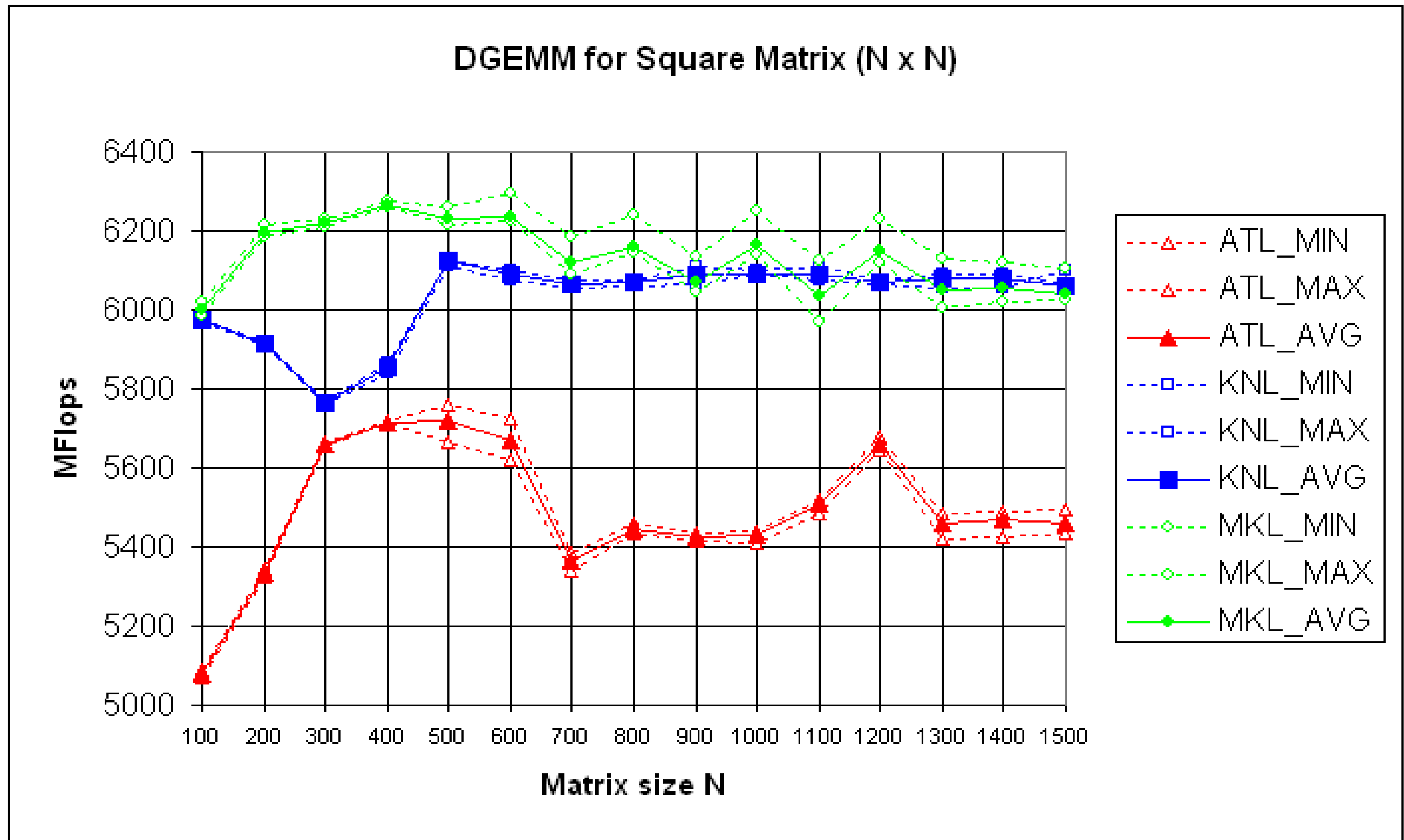
Best decomposition:

- use dotproduct6 kernel
- decompositions found also for degenerate matrix sizes (very thin in one or two dimensions)

The following results are NOT used for the selection of kernels.

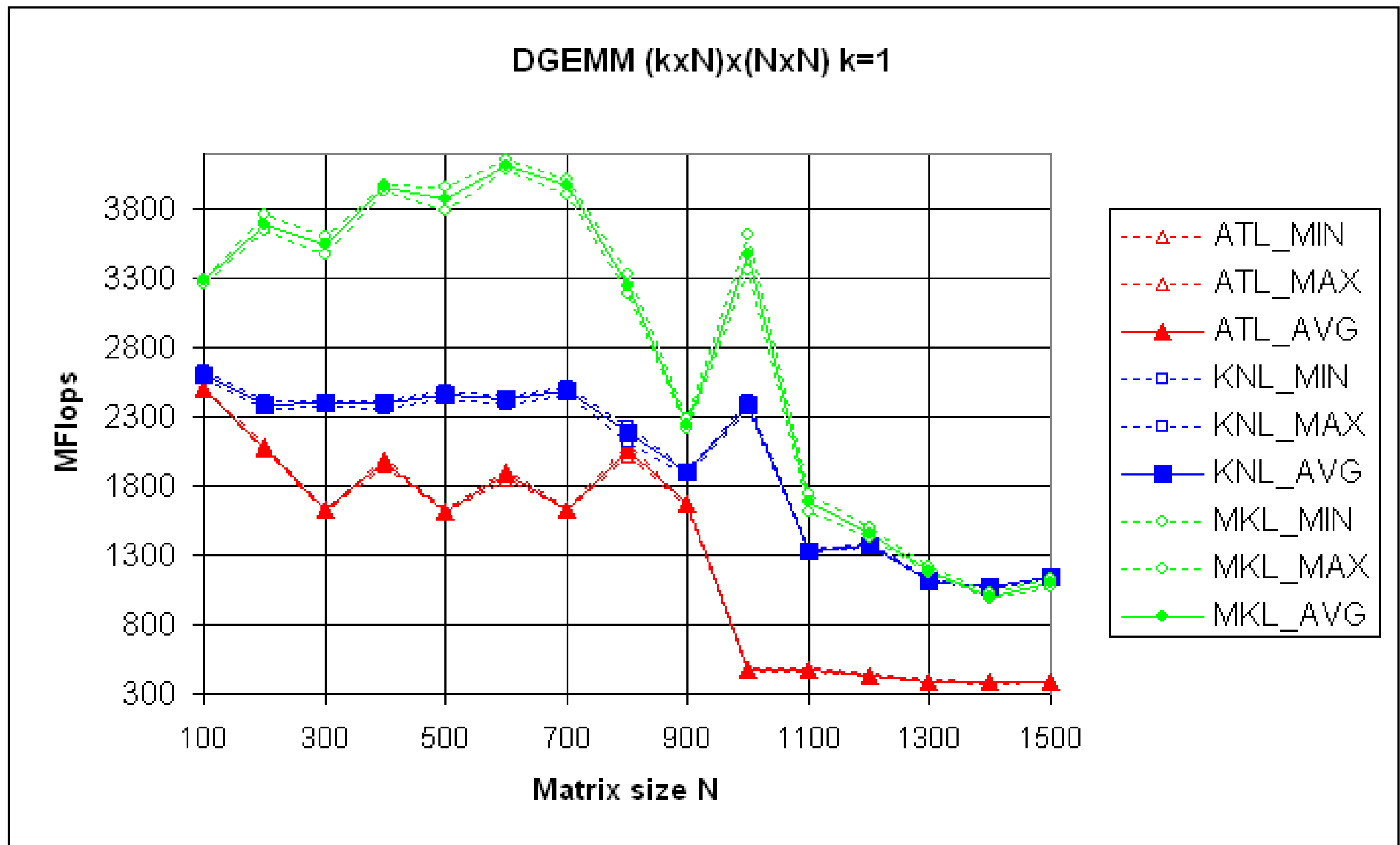
→ They are experimental results of the decompositions.

(NxN) (NxN) DGEMM Performance



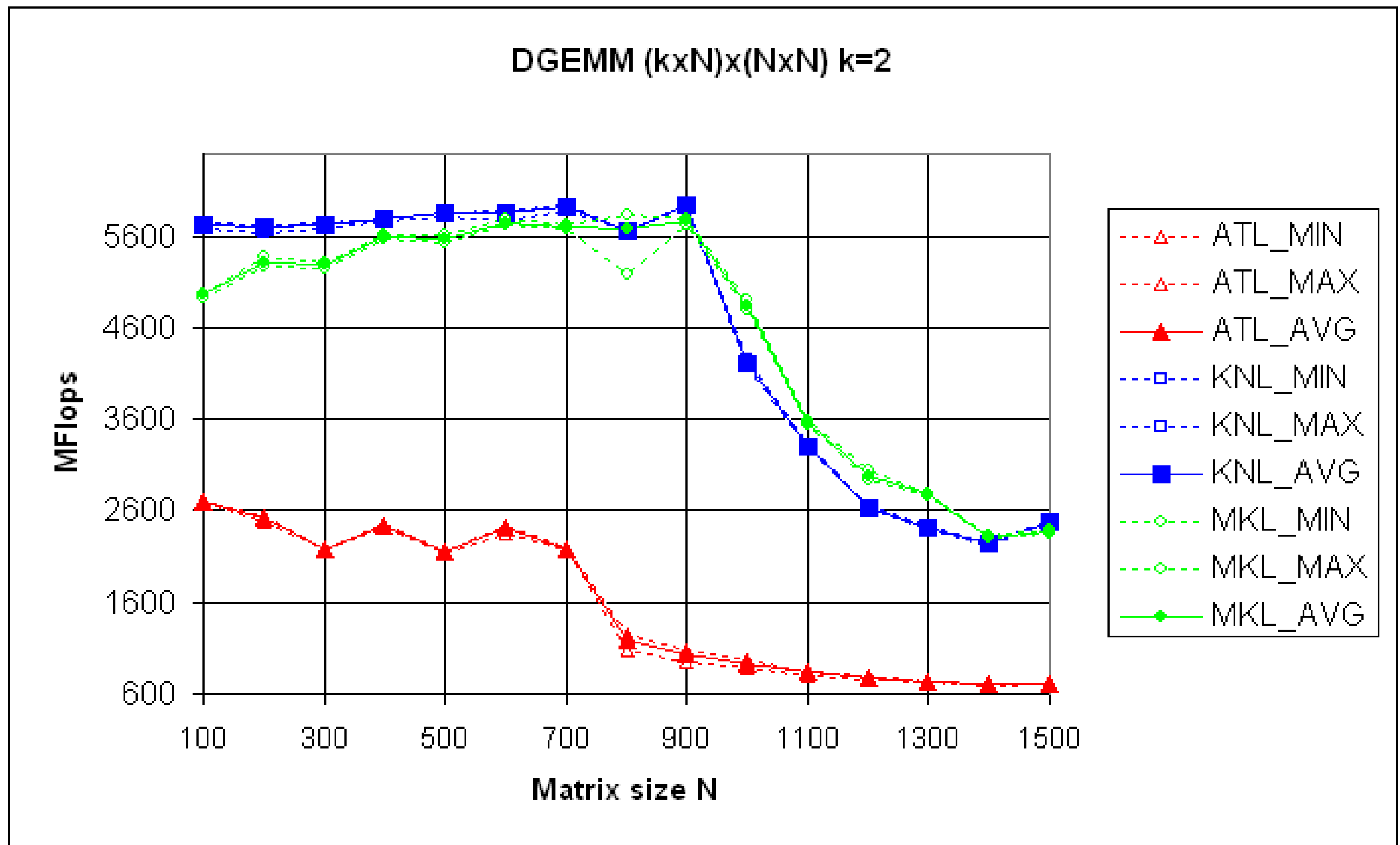
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(1xN)(NxN) DGEMM Performance



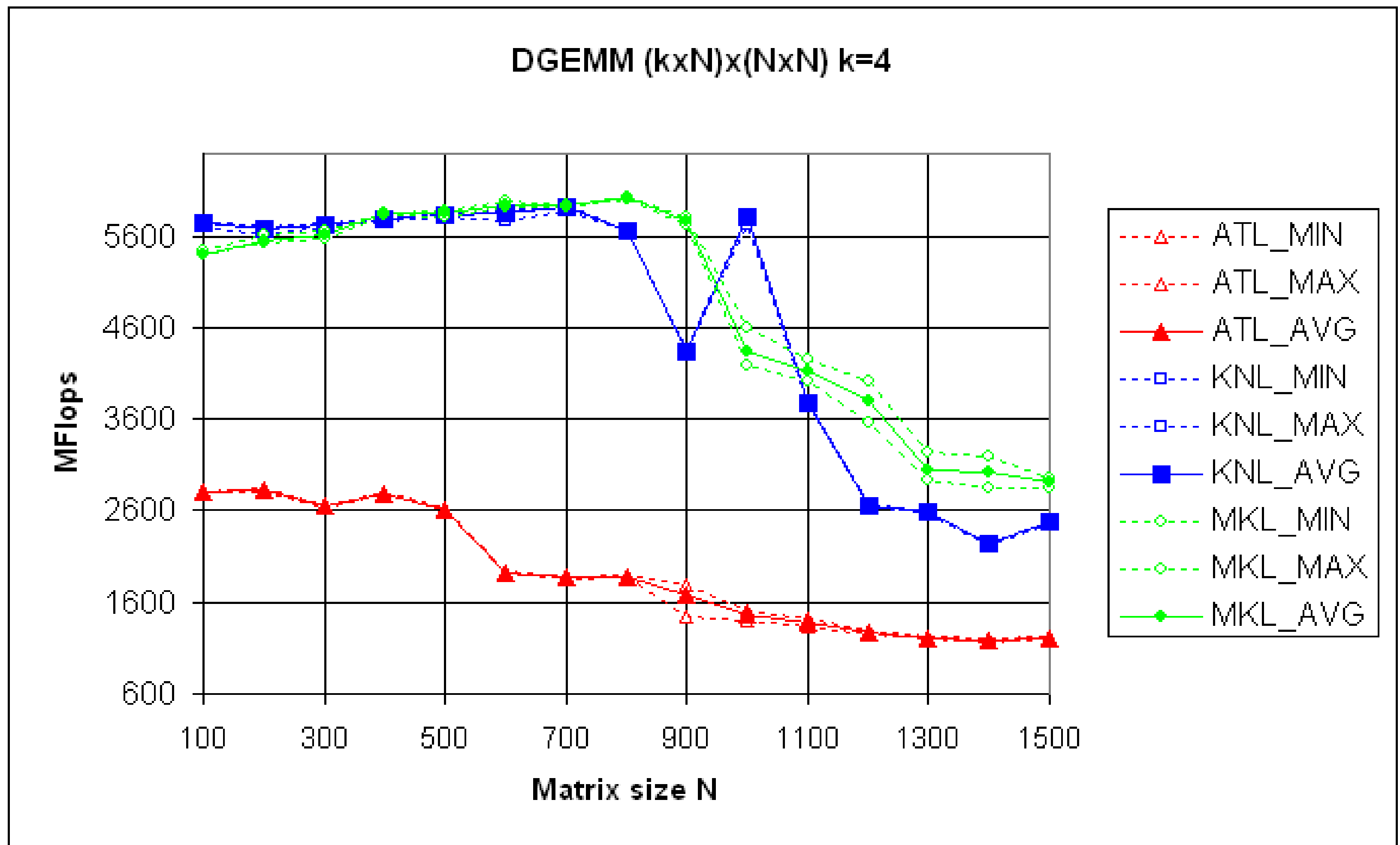
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(2xN)(NxN) DGEMM Performance



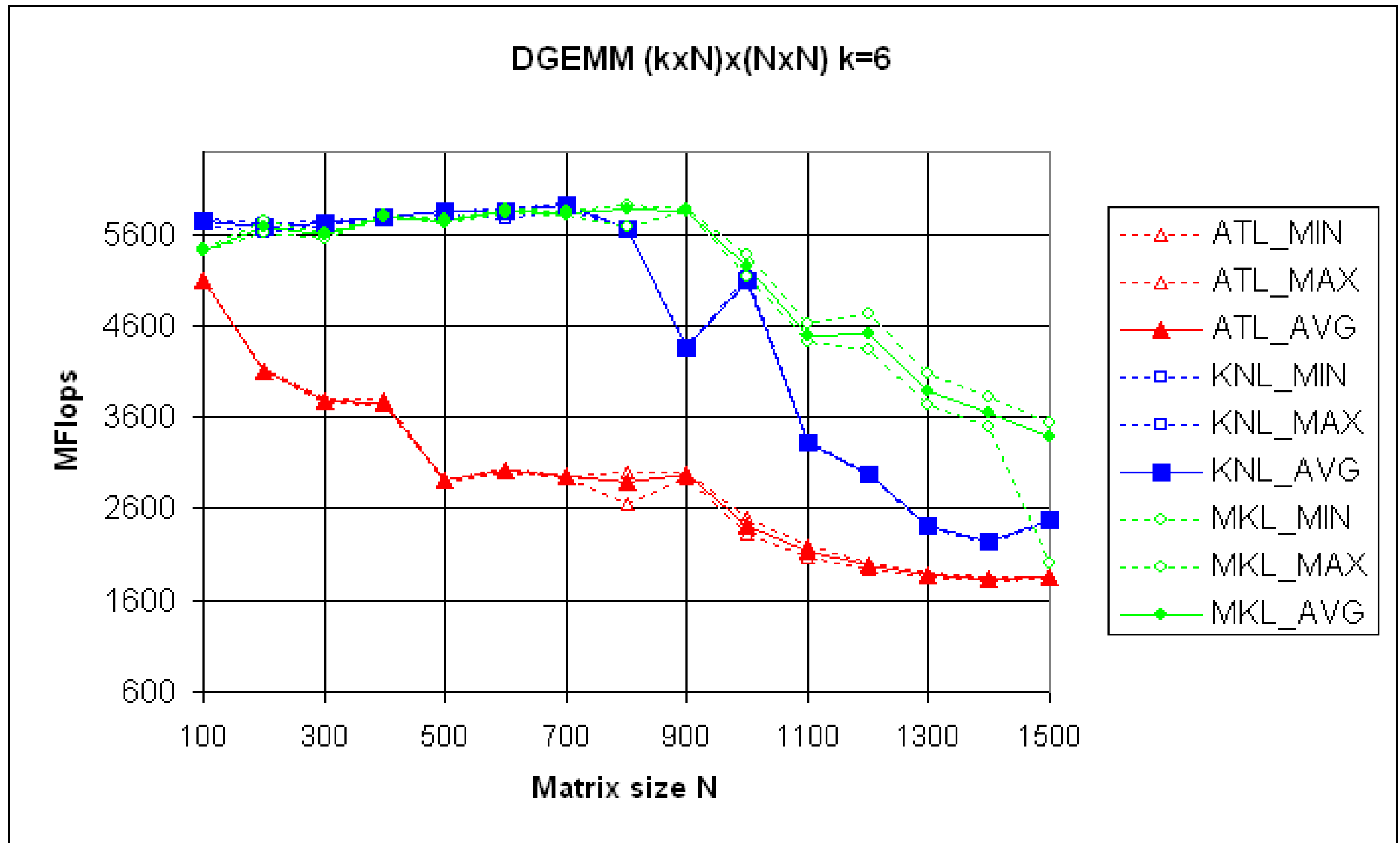
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(4xN)(NxN) DGEMM Performance

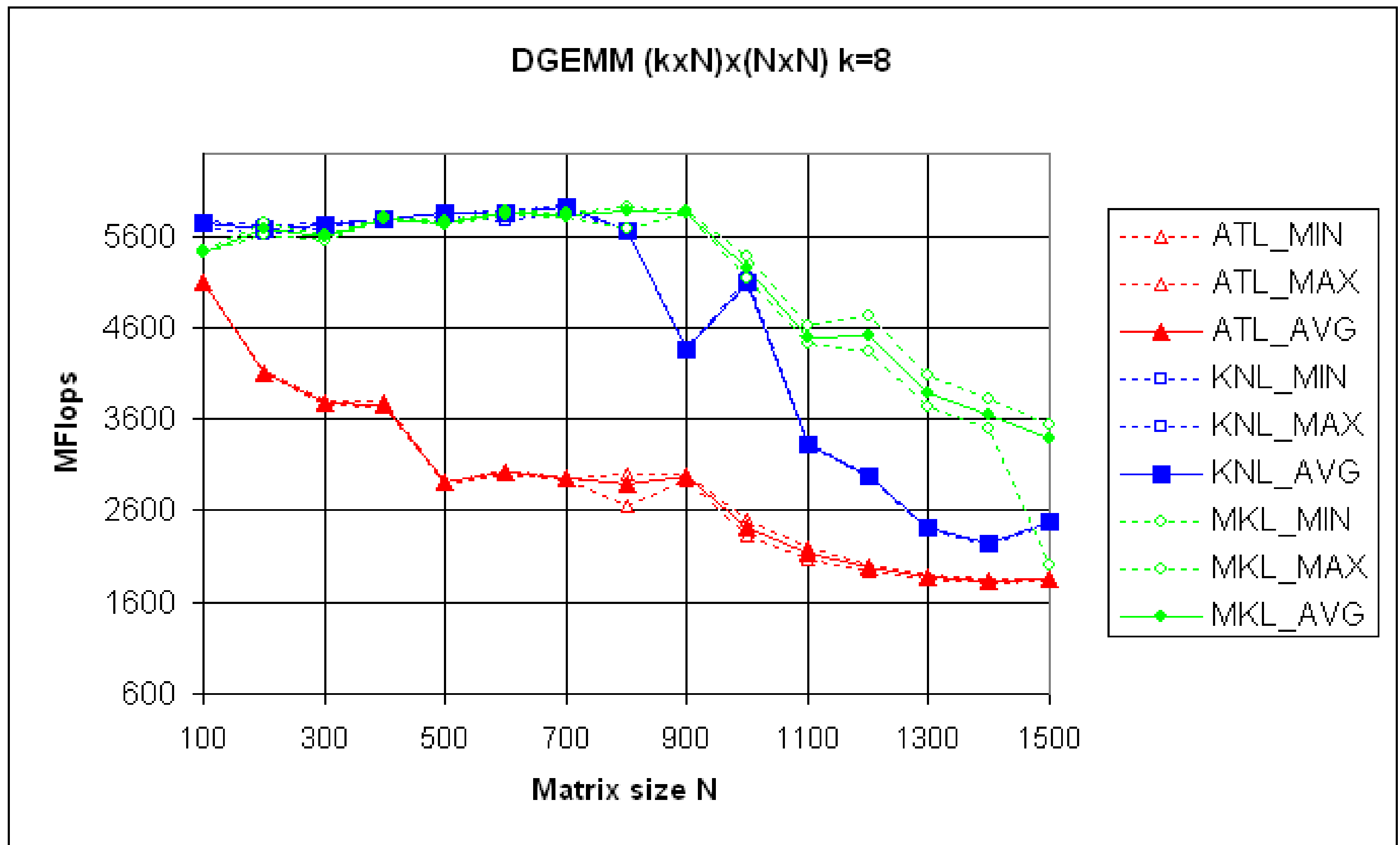


UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(6xN)(NxN) DGEMM Performance

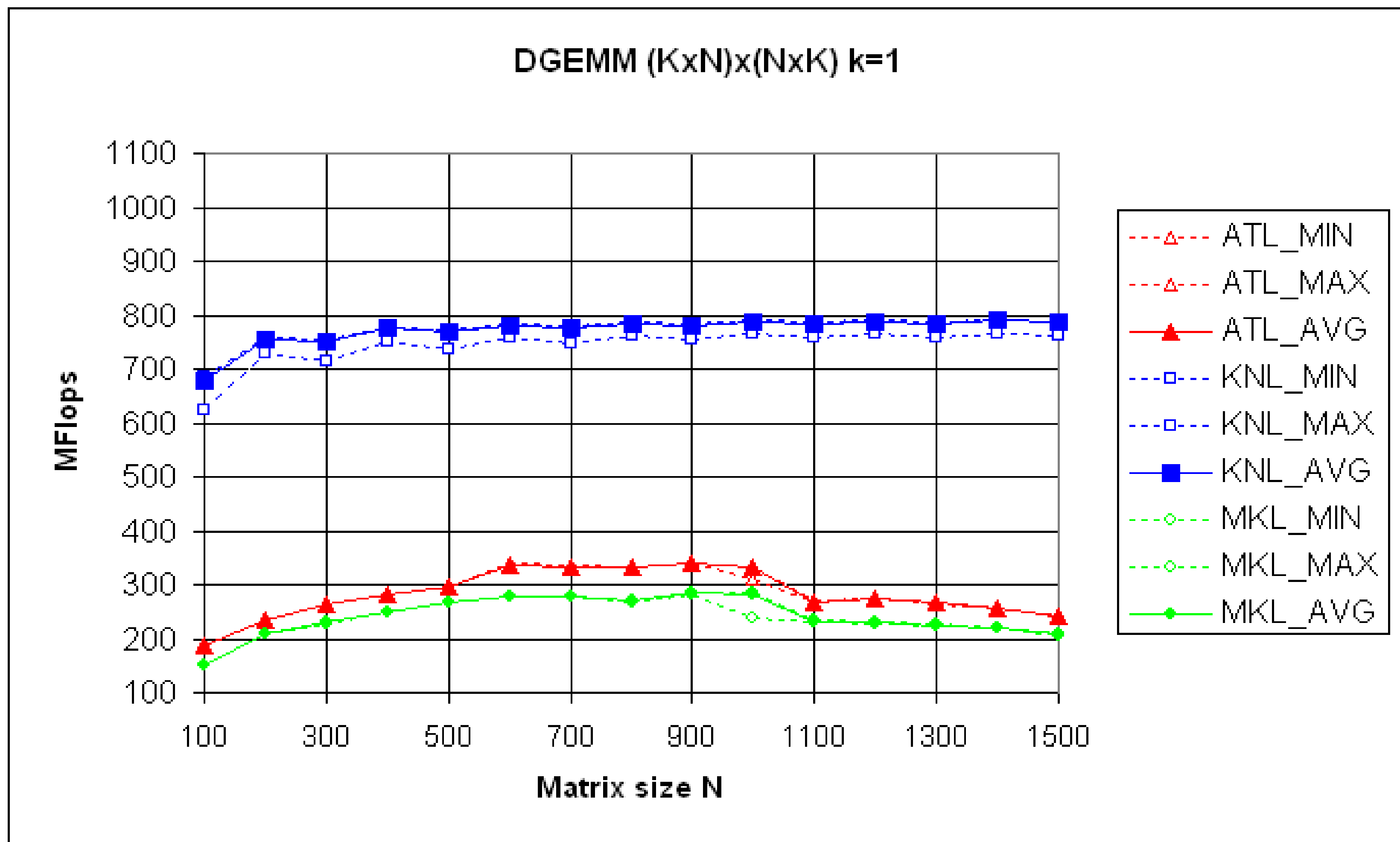


(8xN)(NxN) DGEMM Performance



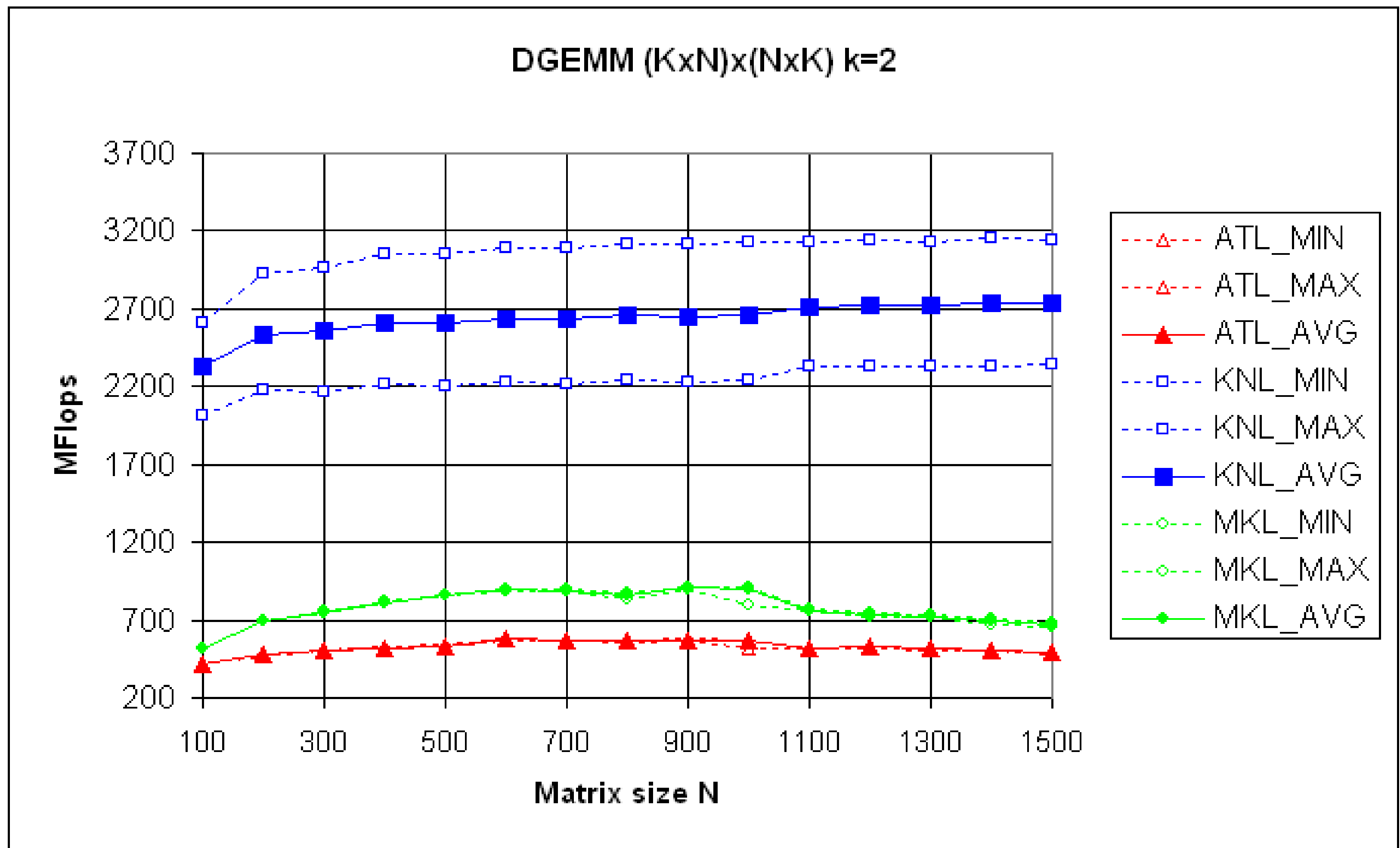
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(1xN)(Nx1) DGEMM Performance



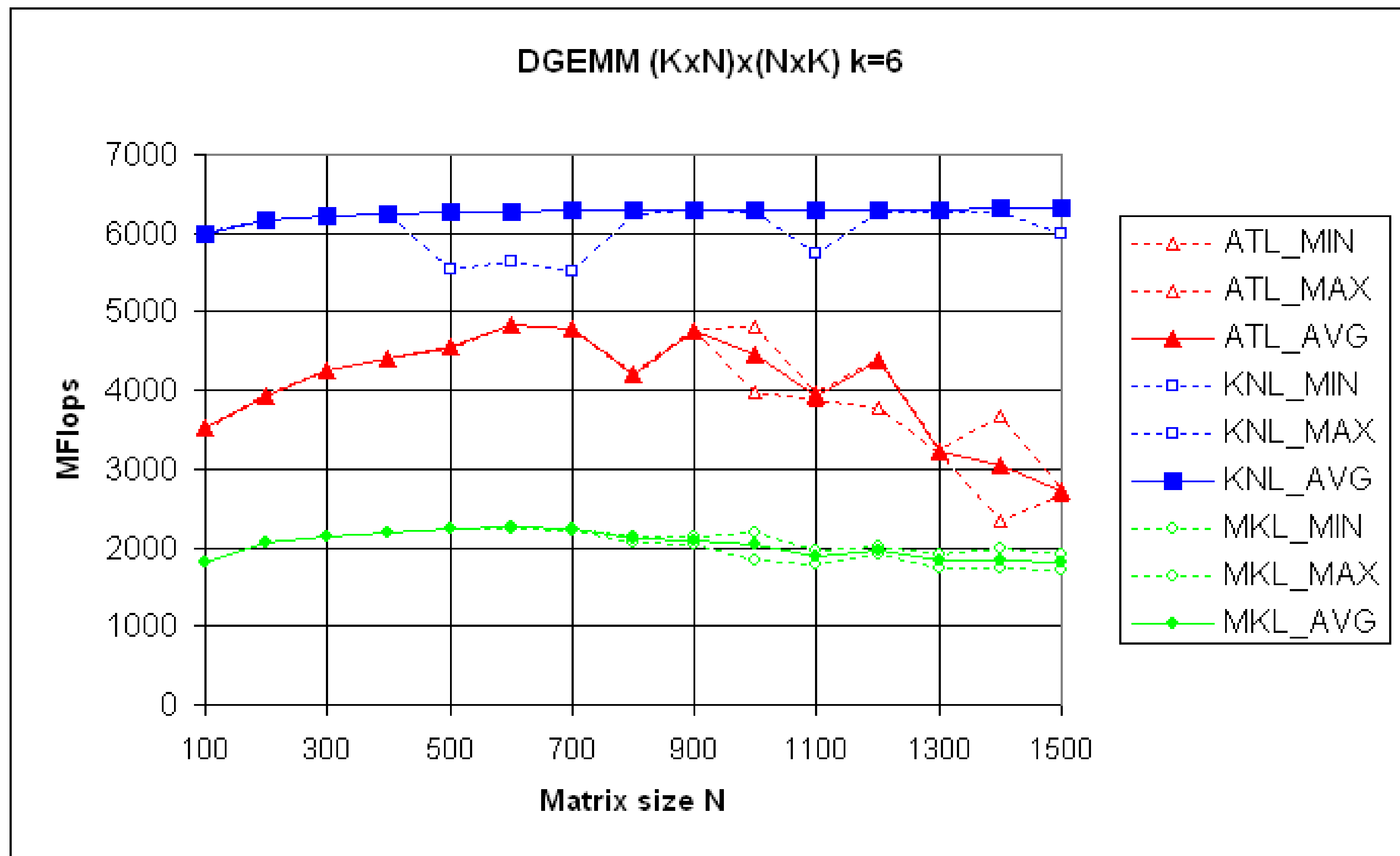
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(2xN)(Nx2) DGEMM Performance



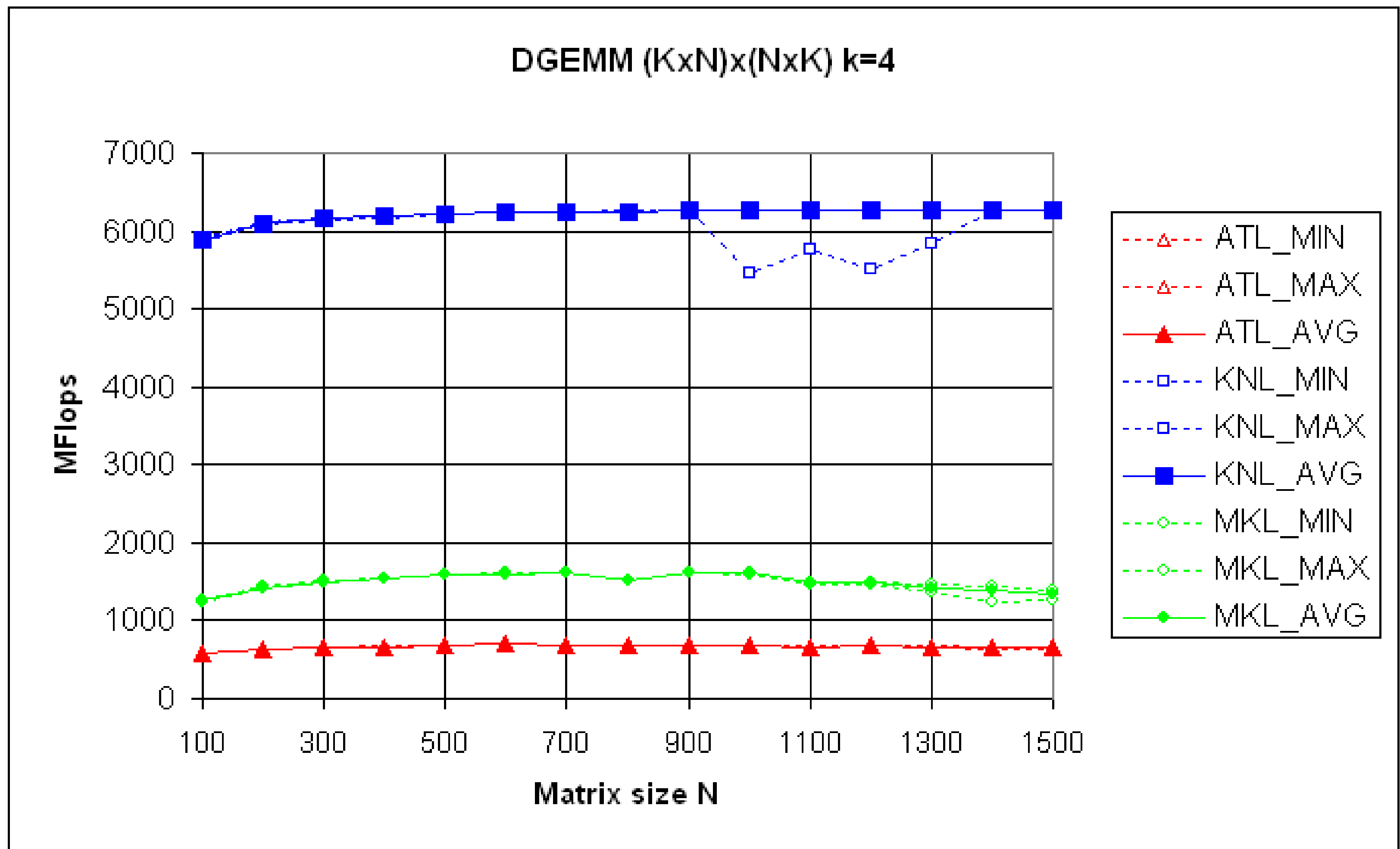
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(6xN)(Nx6) DGEMM Performance



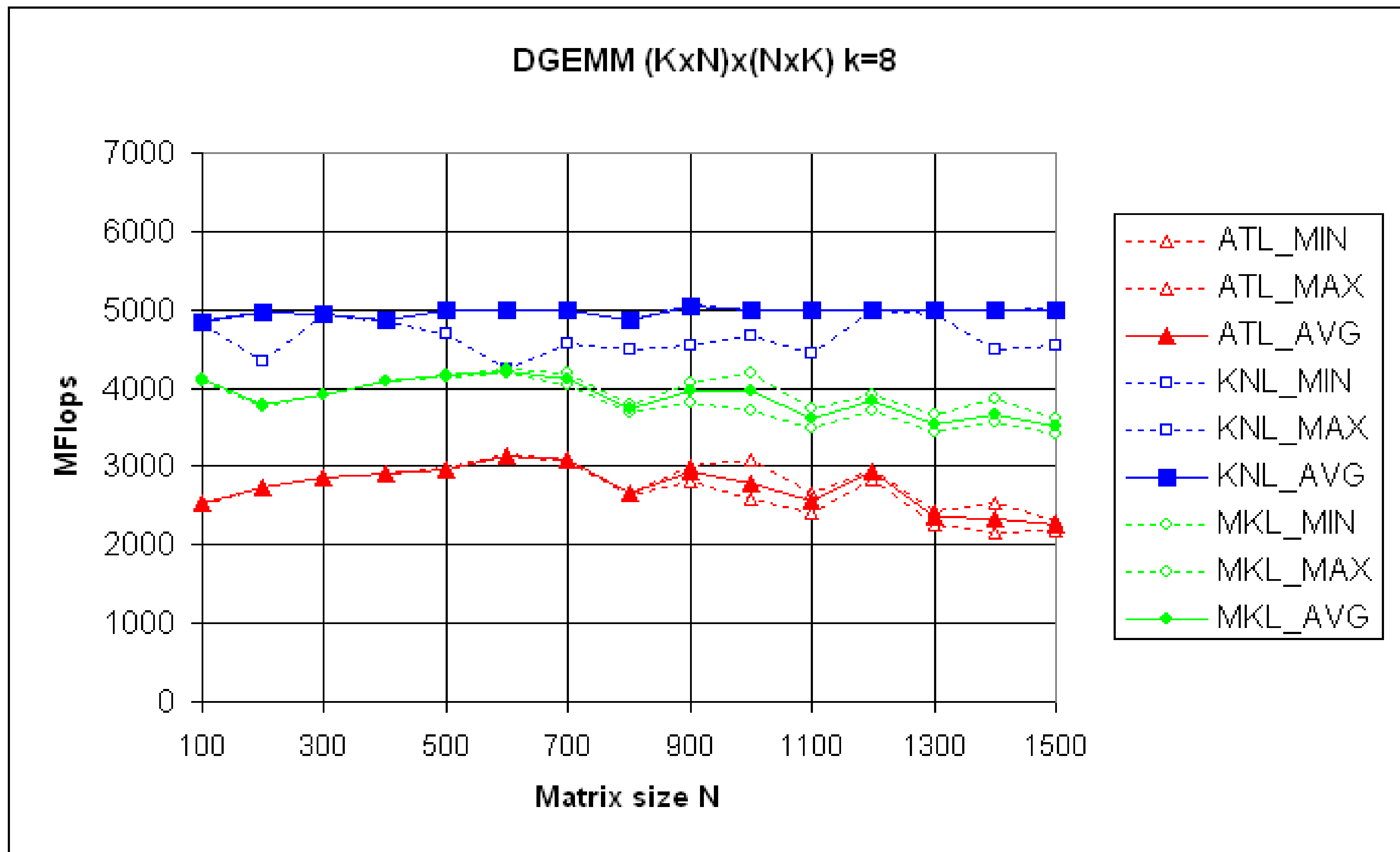
UNIVERSITE DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(4xN)(Nx4) DGEMM Performance



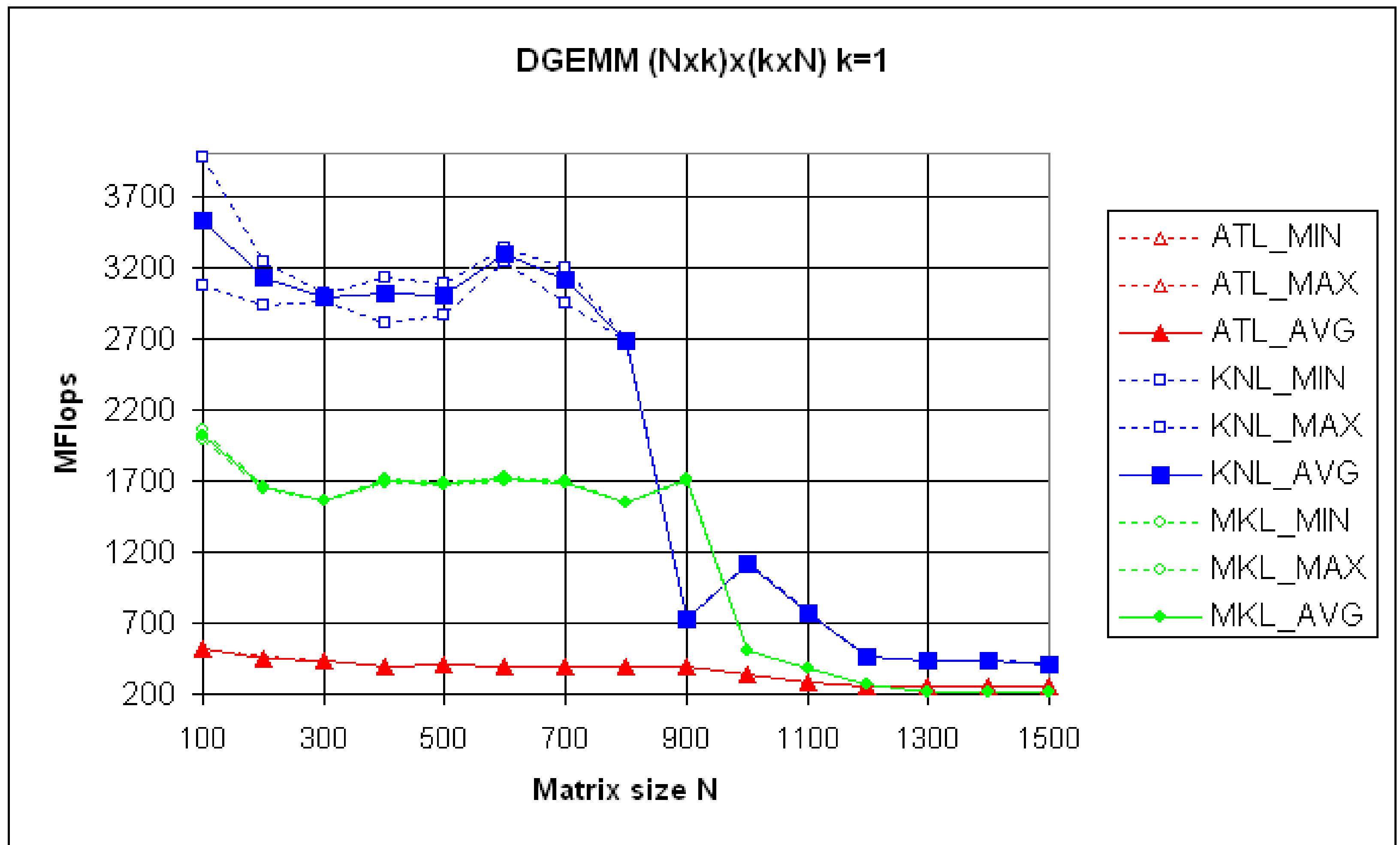
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(8xN)(Nx8) DGEMM Performance



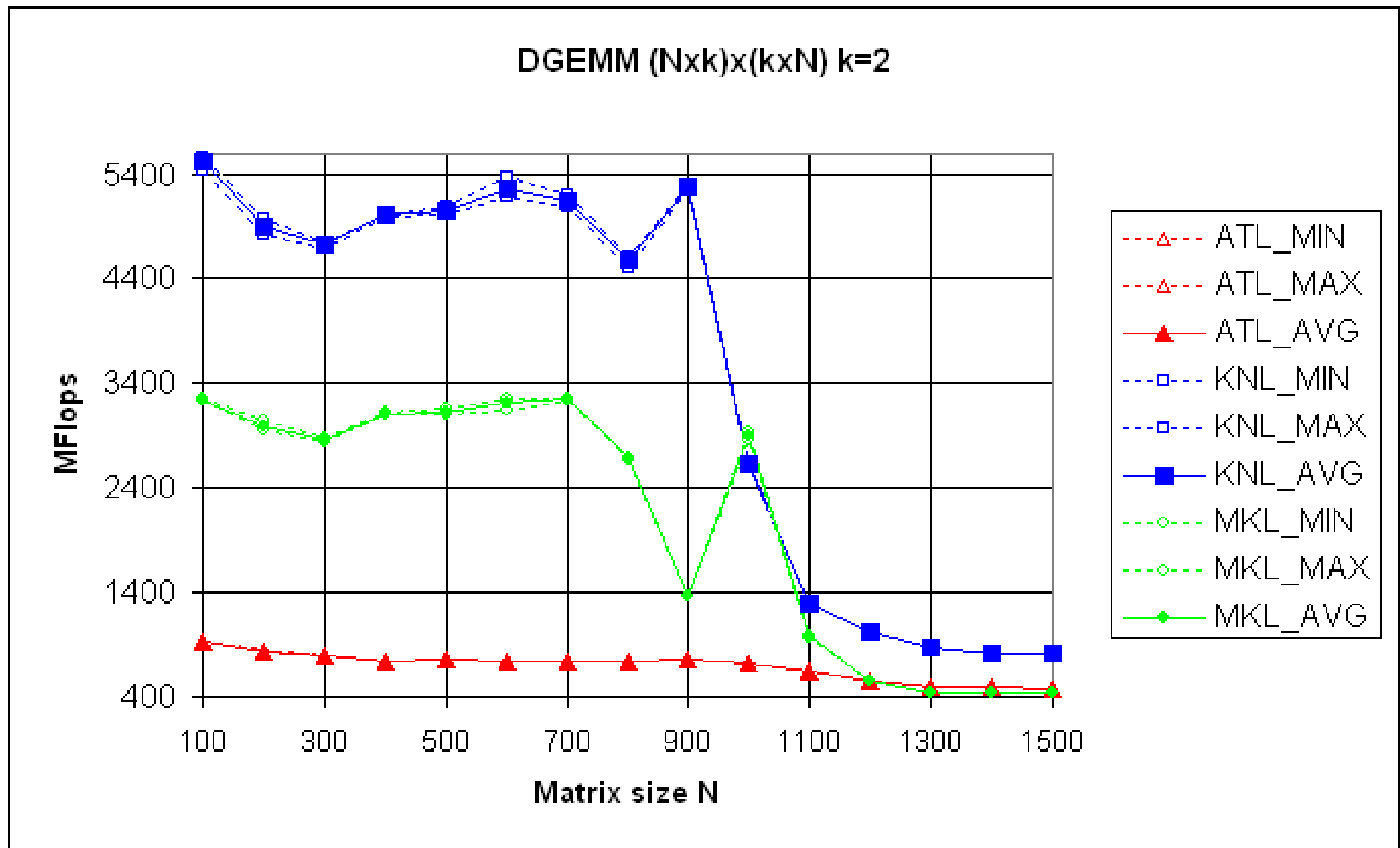
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(Nx1)(1xN) DGEMM Performance



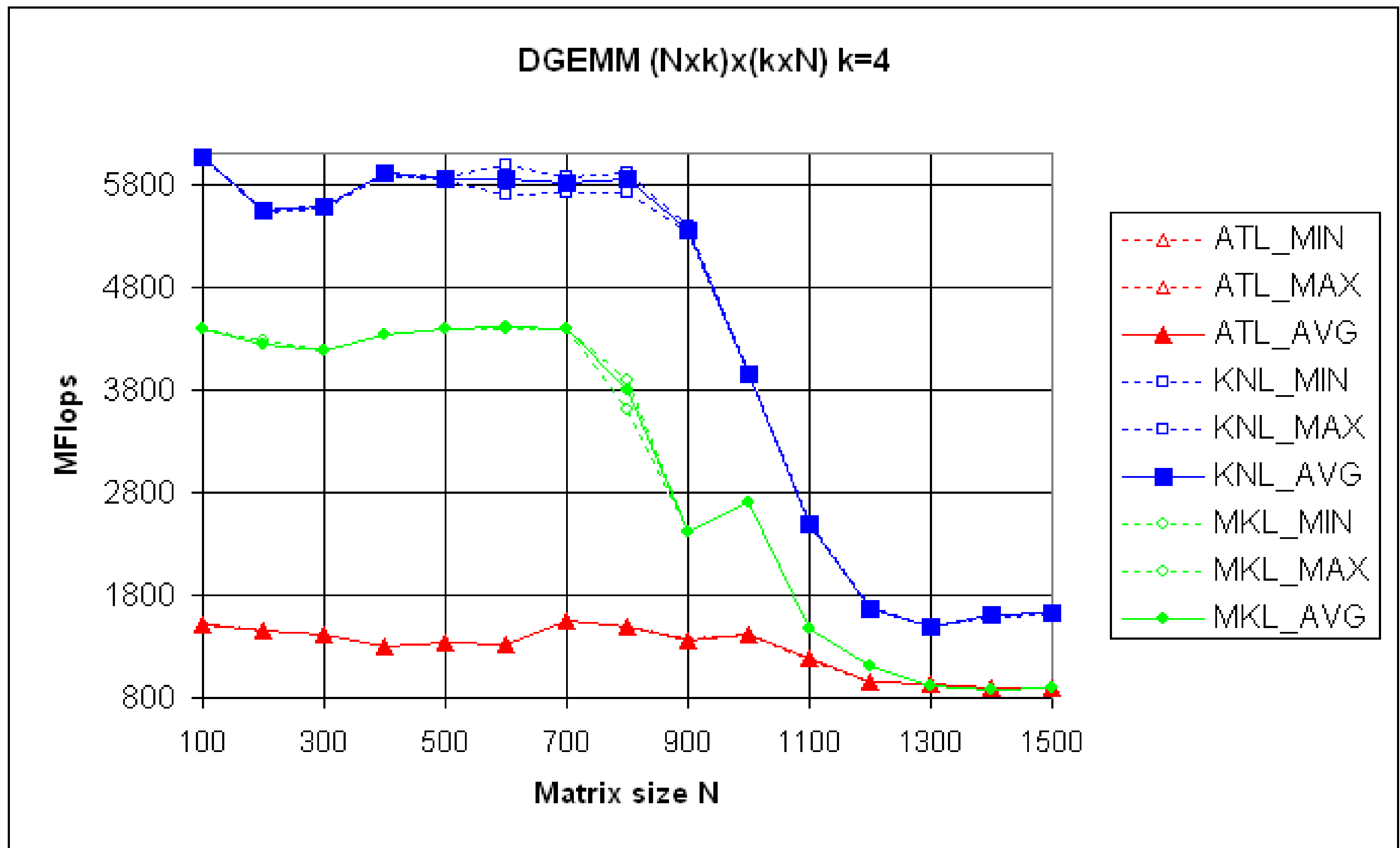
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(Nx2)(2xN) DGEMM Performance

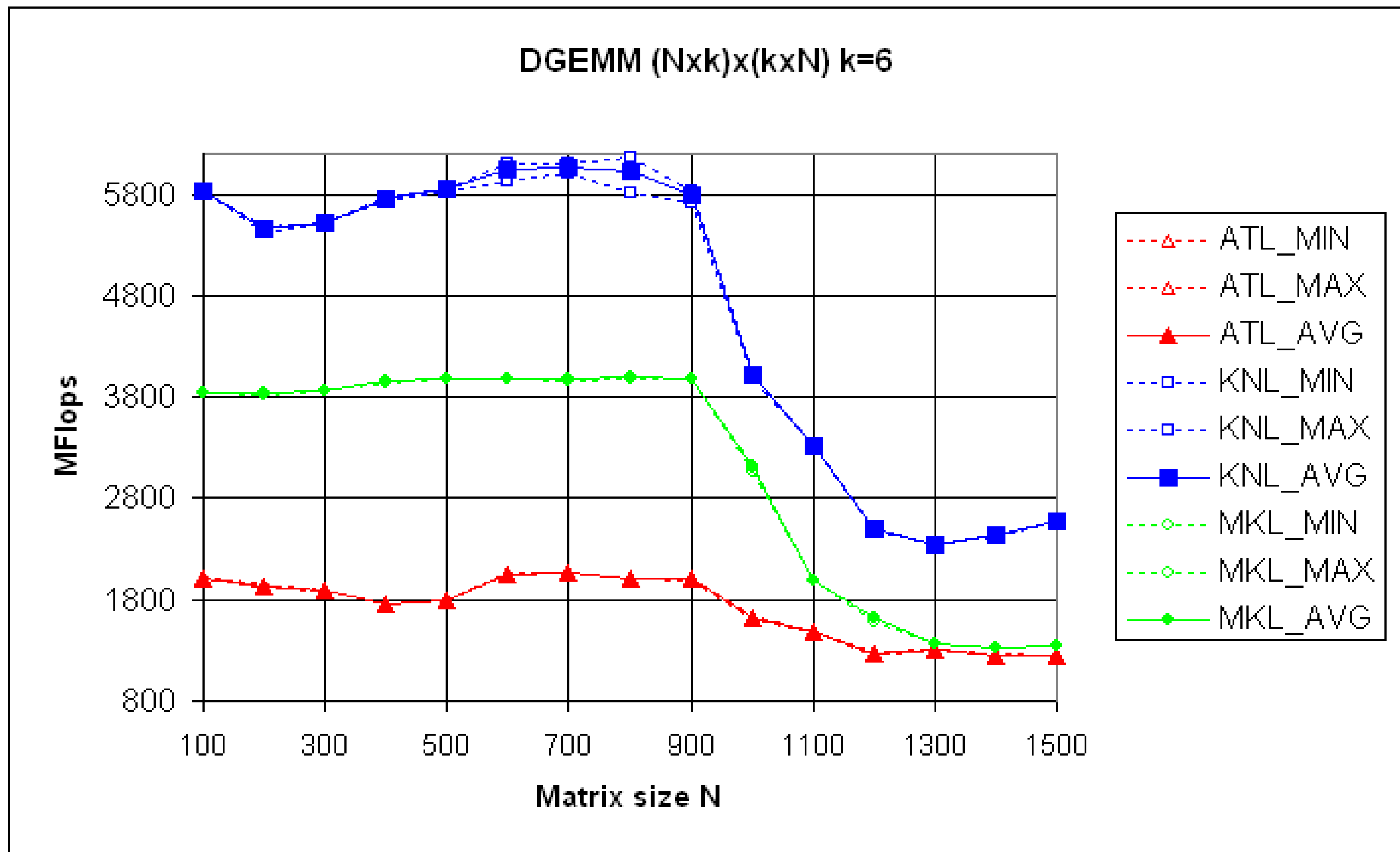


UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(Nx4)(4xN) DGEMM Performance

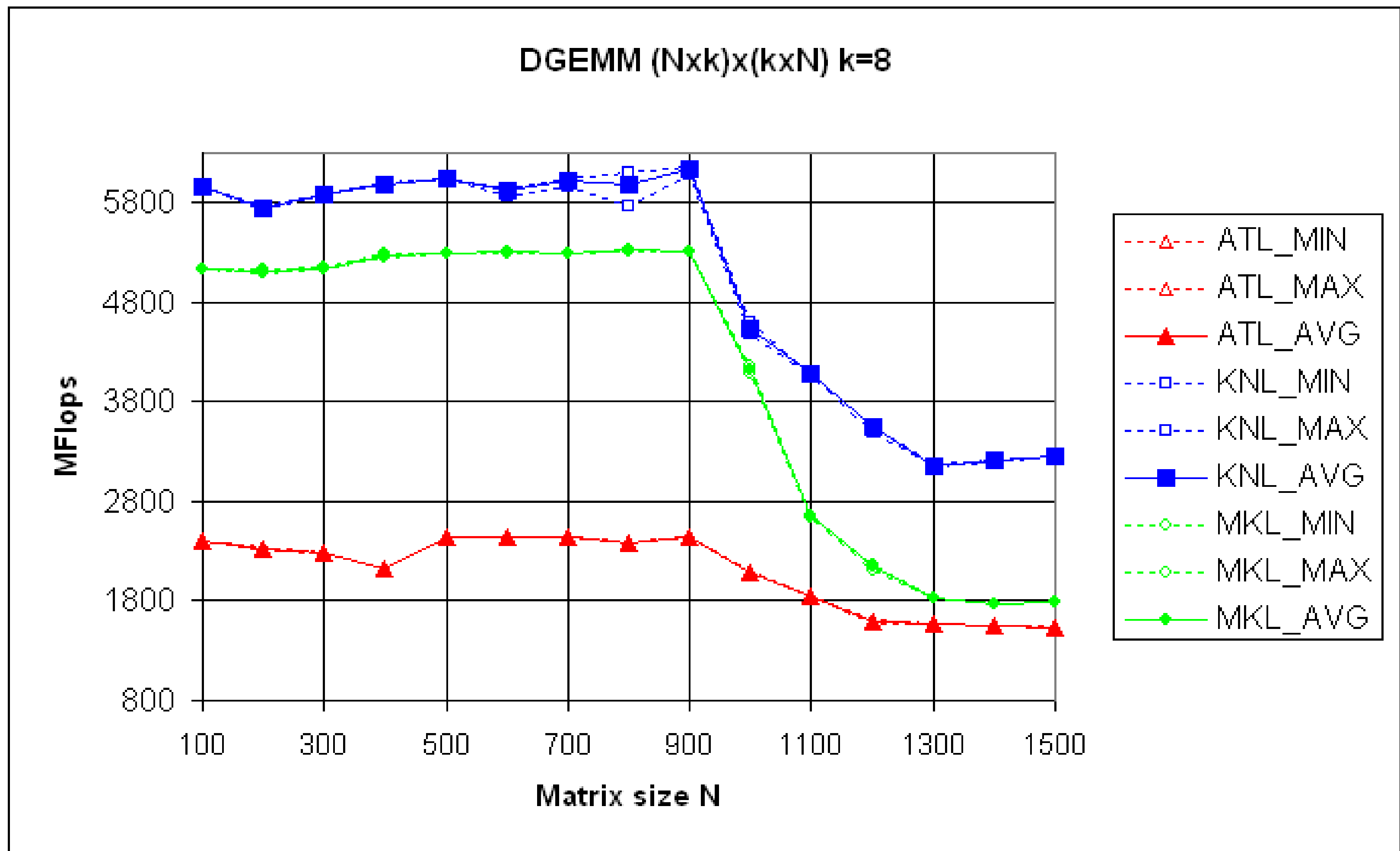


(Nx6)(6xN) DGEMM Performance



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(Nx8)(8xN) DGEMM Performance



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Dealing with L2/L3

Trying to minimize systematically the number of misses is the wrong approach

- What matters is the overall cost:

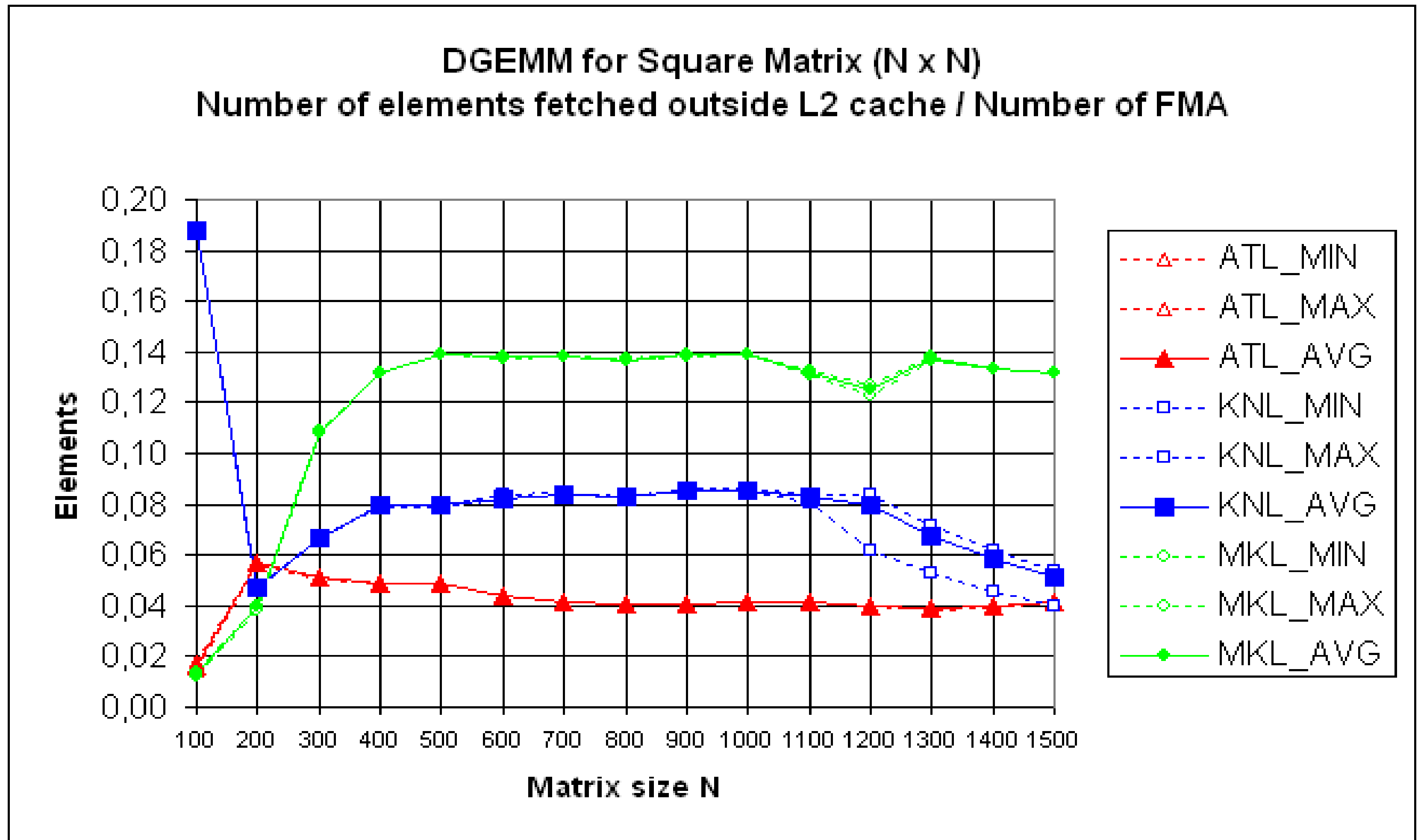
  Σ miss x cost of this miss,

  where the cost is the bubbles generated

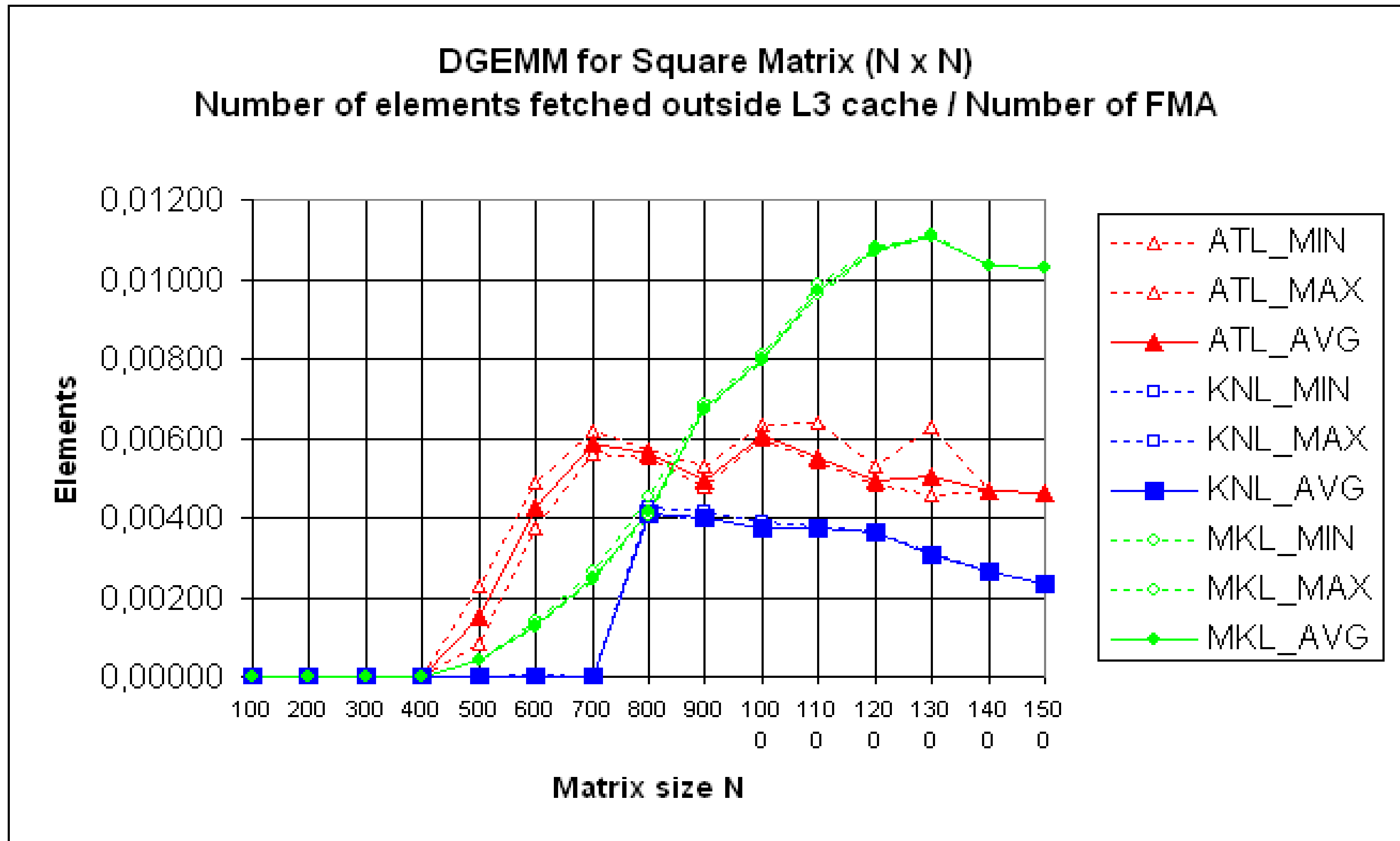
- Itanium has an interesting L3: for example computing a dot product for which one array is in L2, while the second one is in L3 is as fast as if the two operands are in L2

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

(NxN)(NxN) DGEMM L2 Behavior



DGEMM (NxN) (NxN) L3 Behavior

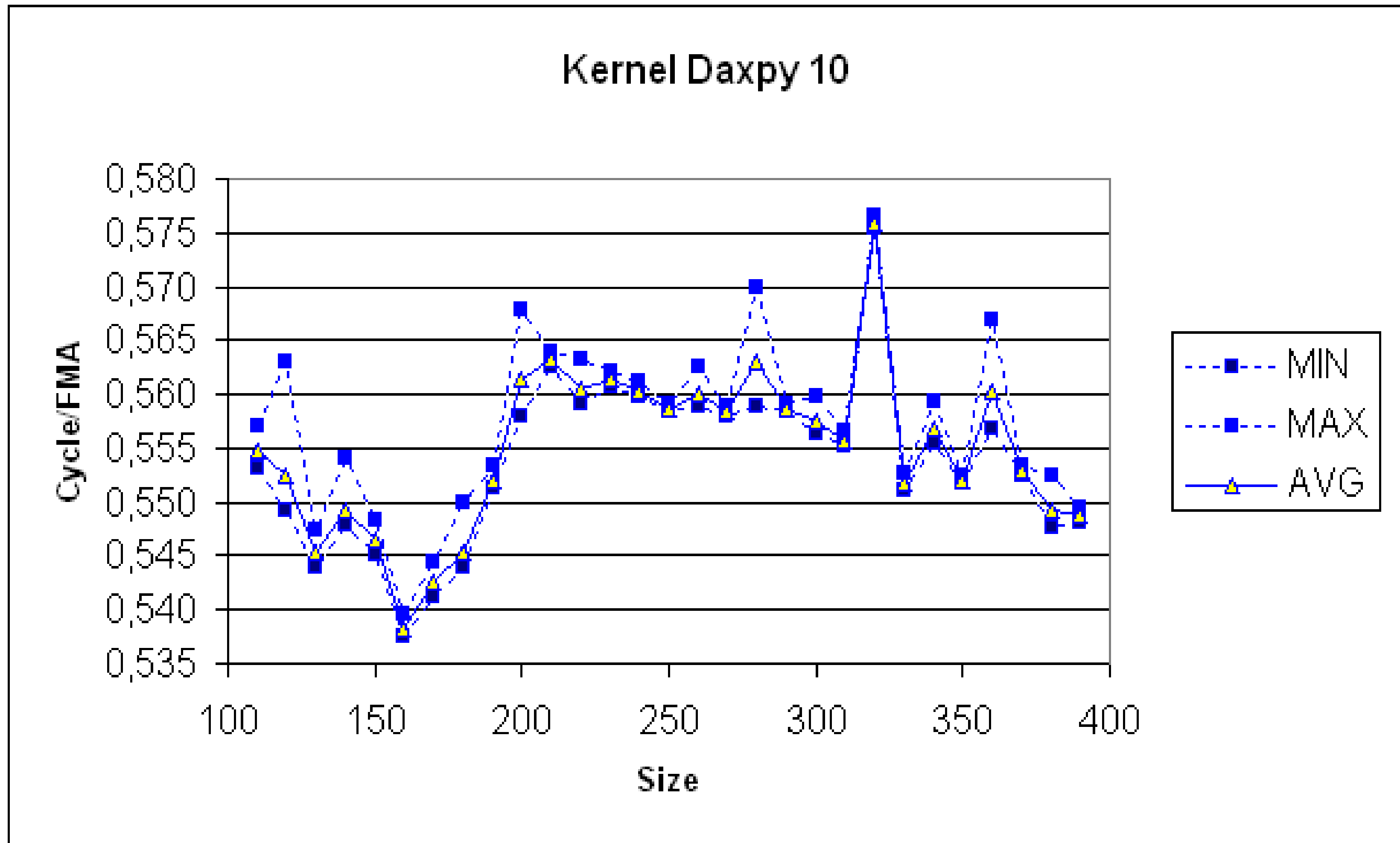


Stability issues

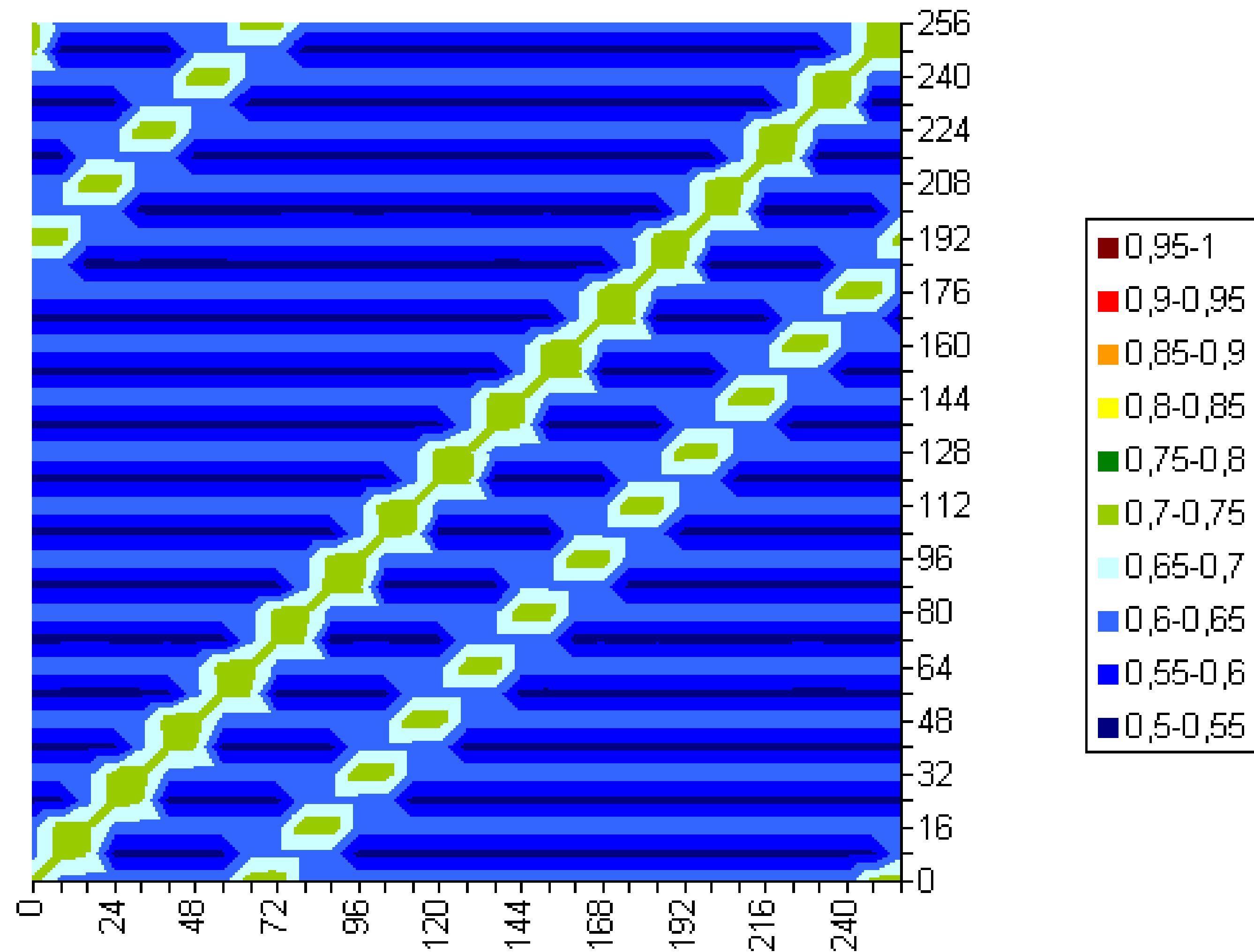
- **Performance are unstable**
  - bank conflicts
  - Load Store Queue problems
- SOLUTION: use the copy operation before each kernel

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Daxpy10 Stability issues

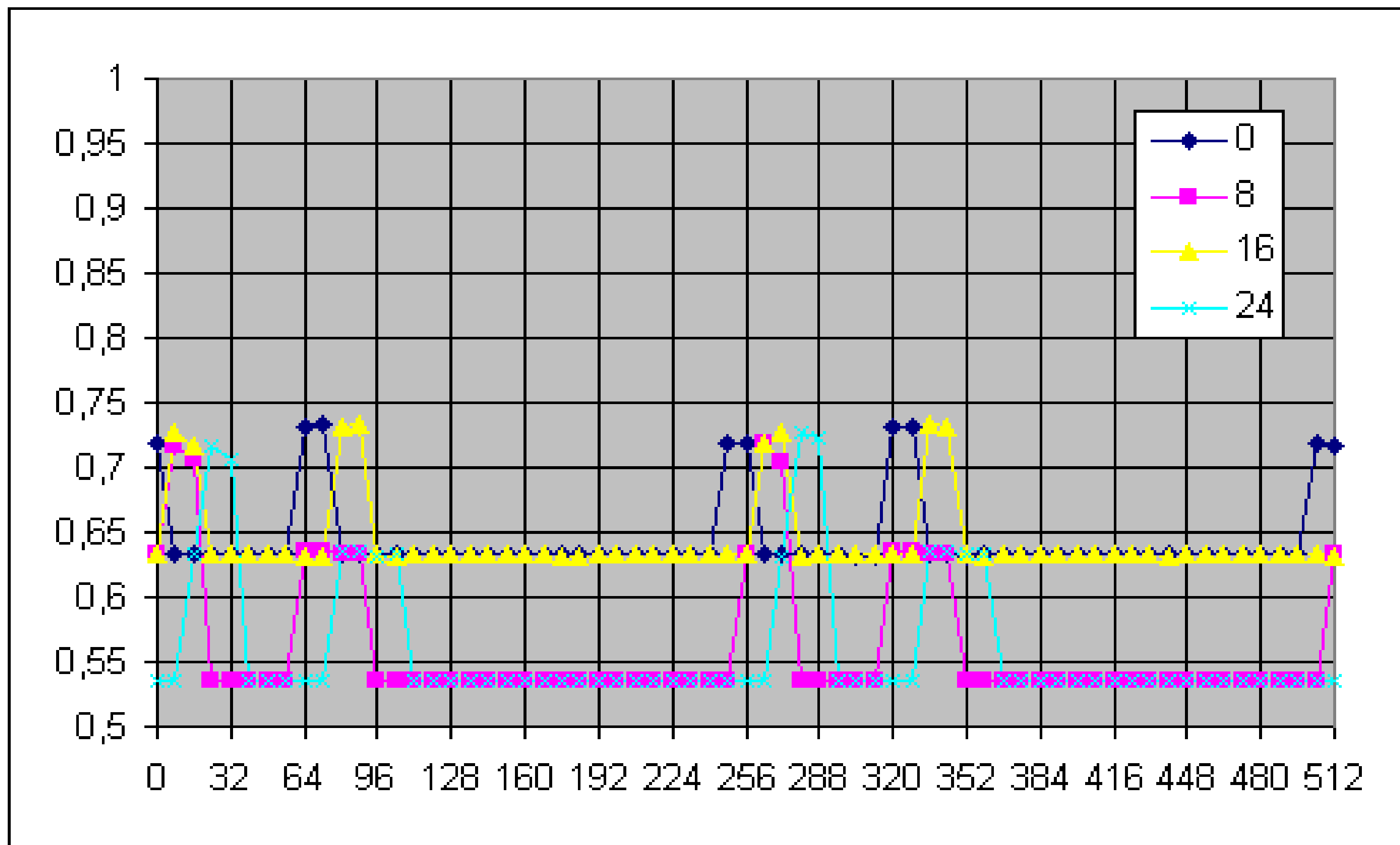


Impact of Array Offset on Daxpy10 U2 N = 200



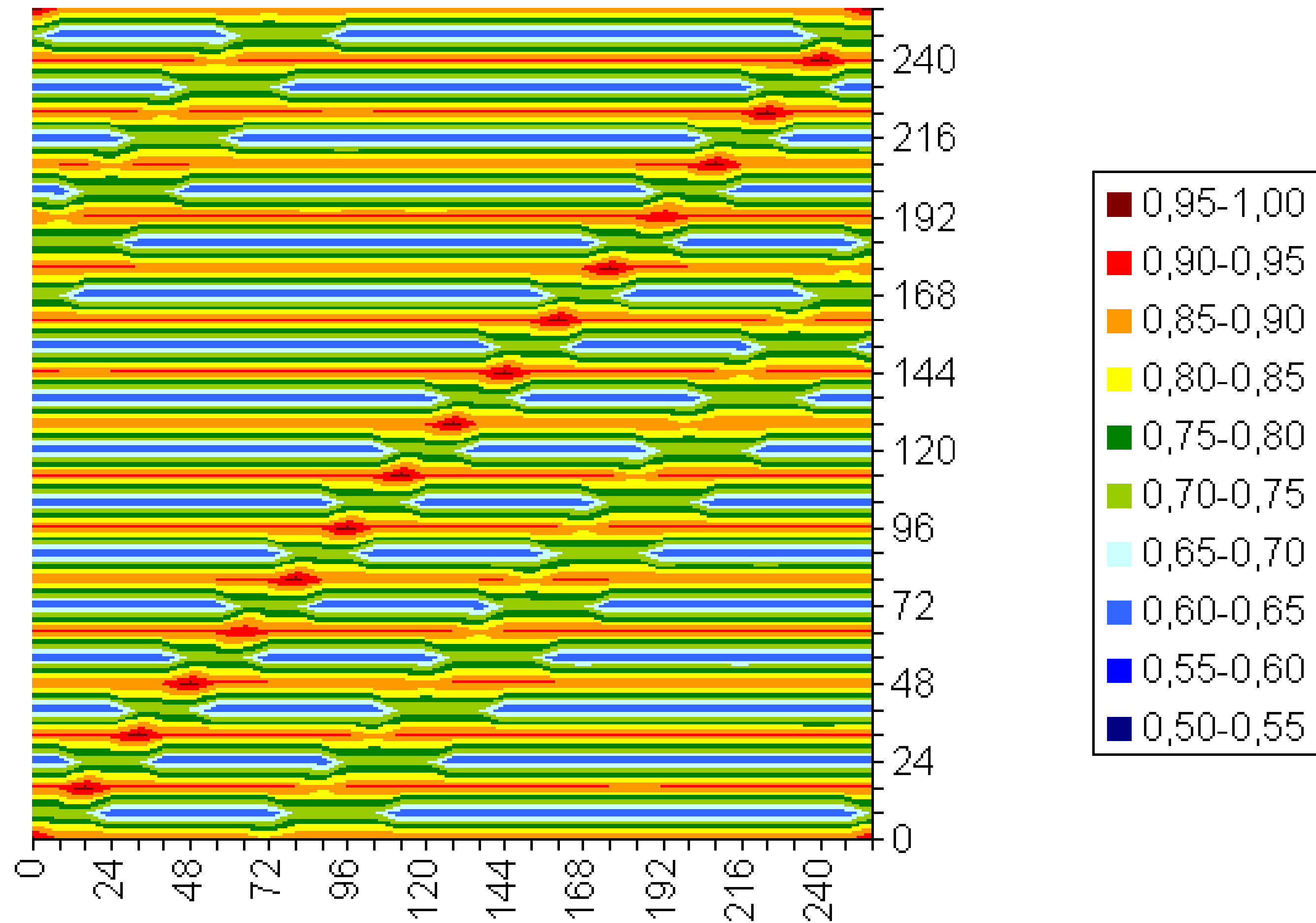
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Impact of Array Offset on Daxpy10 U2 (N=200)



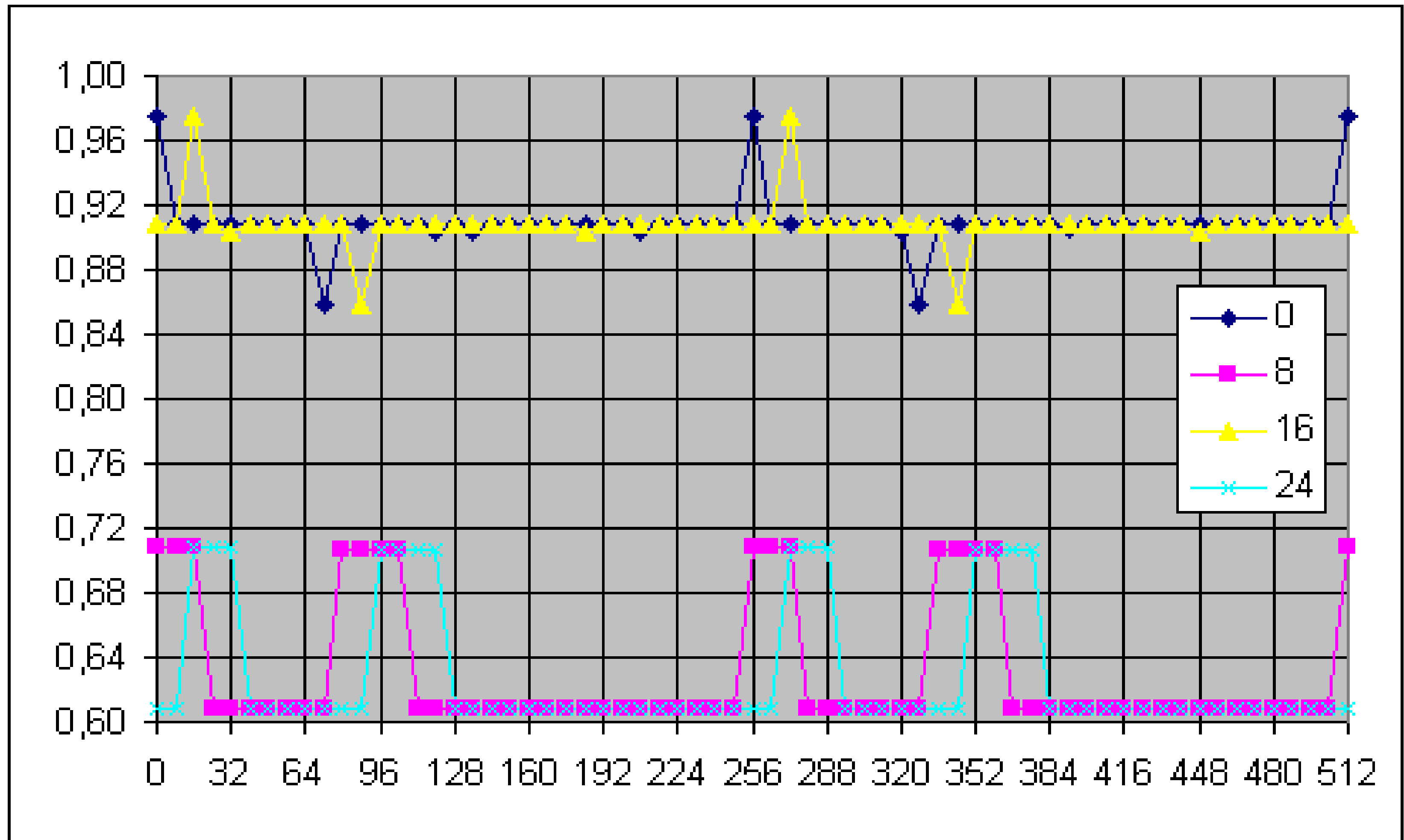
UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Impact of Array Offset on Daxpy10 U2 (N=800)



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Impact of Array Offset on Daxpy10 U2 (N=800)



UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Sum up for kernel decomposition

A method to generate efficient codes with source to source transformations

- Search is limited to kernels
- → Seems necessary due to complexity of hardware mechanisms at this level
- Low level optimizations taken care of by the compiler
- Still need for more complex models to reduce high level optimization search

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Perspectives

- Extend the approach to other library codes (banded case, finite difference codes)
- Apply same approach to application loop nests
- First results on Pentium confirm those on Itanium (with SSE vectorization)
- More complex high level transformations needed, as well as a performance model...

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Outline

- Exploring optimization space
  - Manually: Multistage compilation languages
  - Automatically: by search or with a model
  - Combine: Kernel decomposition
- **Feed-back guided code tuning**

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Tools for code tuning

- Performance code tuning
  - Vtune, HPCView, MAQAO...
- Why not use the expertise of these tools as feed back to limit search ?
  - first step: Ocean project, in the 90's
  - nature of feed back:
    - assessment of performance bottle neck
    - identify the limits of the compiler
    - propose optimization paths to improve performance

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

MAQAO: a modular assembly code quality analyzer and optimizer

- Analysis of assembly codes
  - structures (loops, call graph,...)
  - static performance model (compiler and resource bound)
  - identify poor performance patterns
  - find optimizations applied (versioning, inlining, unrolling,...)
- Instrumentation and analyses
  - compute hotpath, value profiling (prefetch distances, bank conflicts, opportunities for versioning)

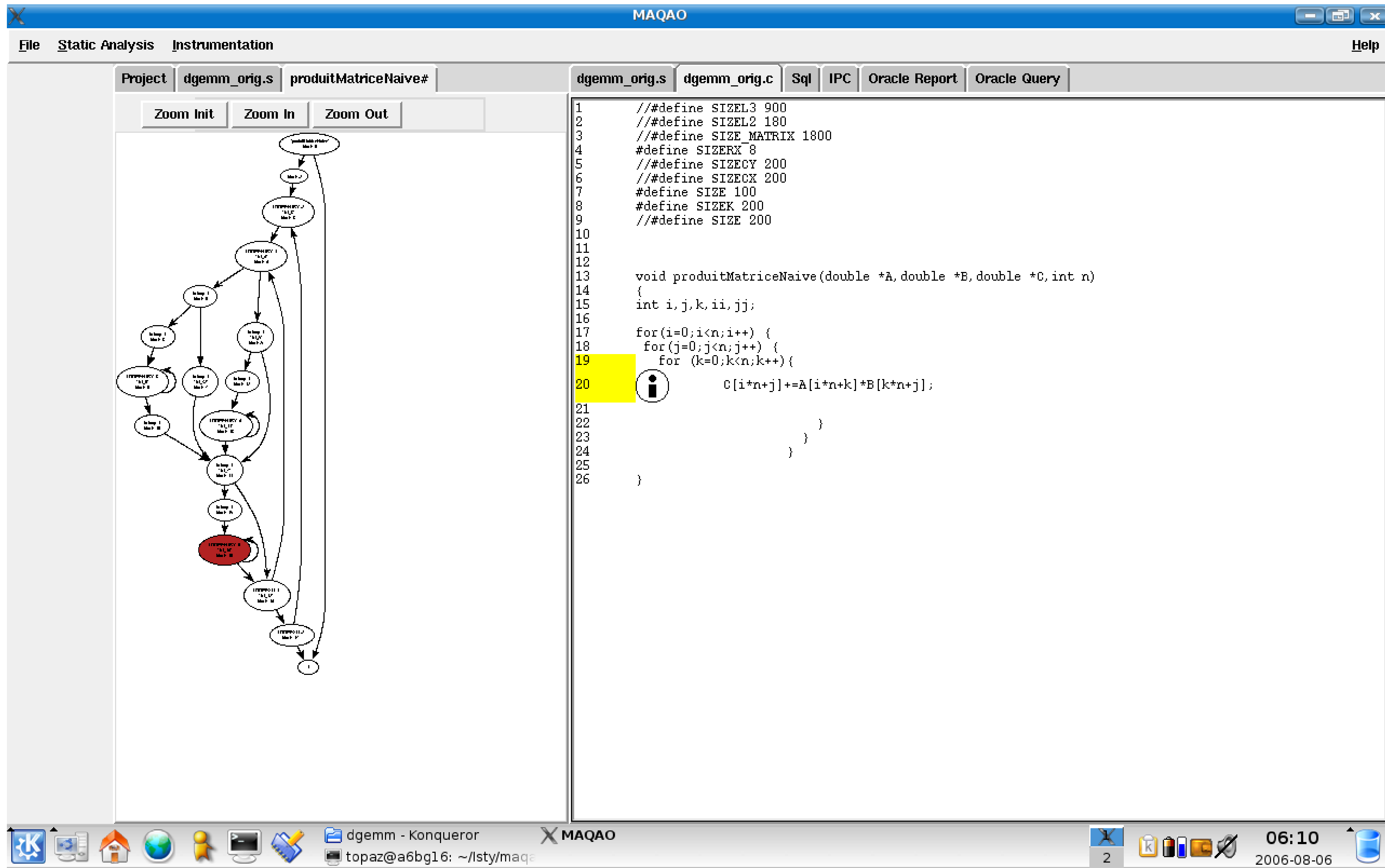
UNIVERSITE DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

Feed back for iterative compilation

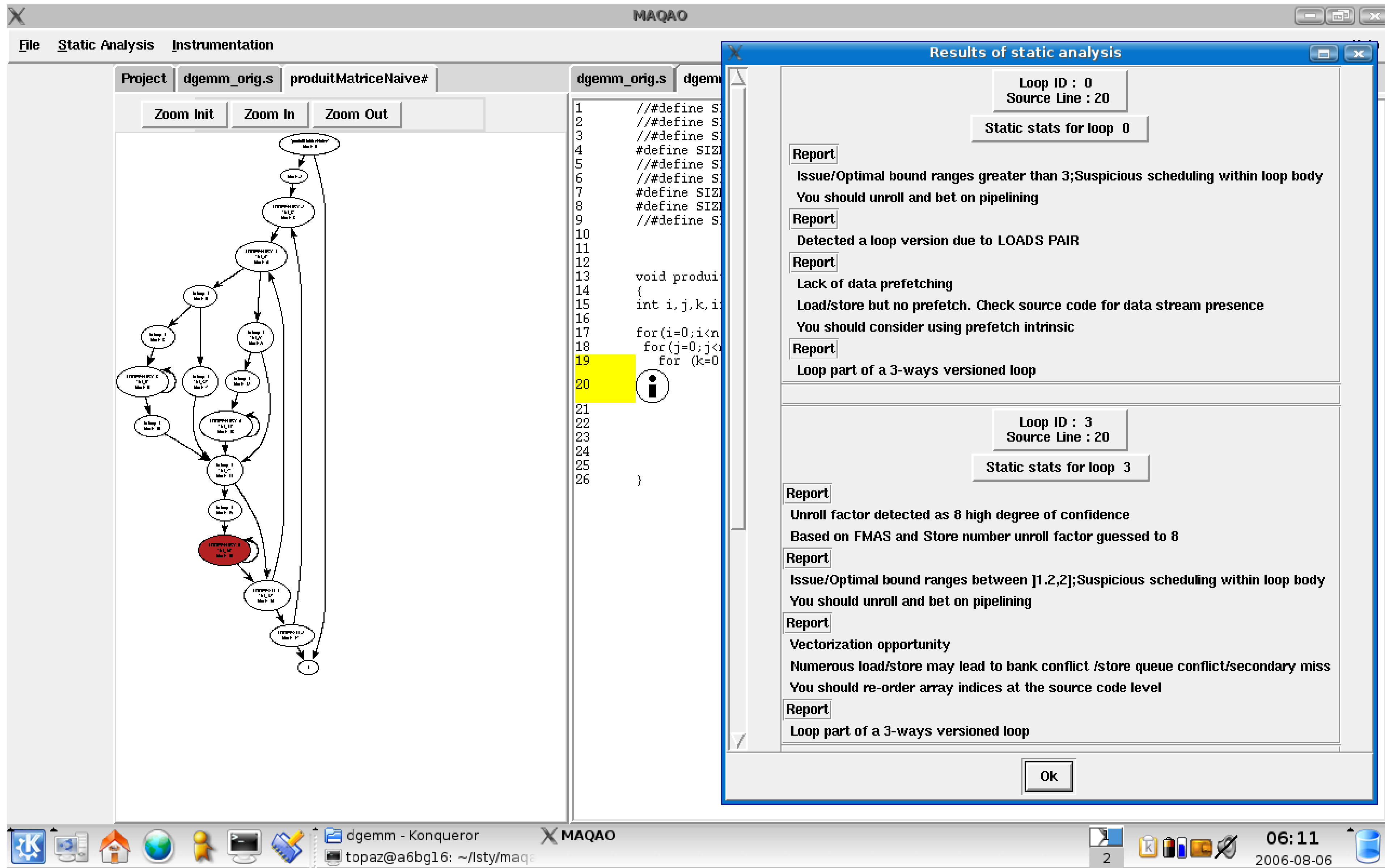
- No execution needed
  - static analysis of assembly performance can avoid some executions
  - comparison between assemblies
- Guiding the search
  - proposing modifications (pragmas or compiler flags)
  - identifying when the code hits the limits of the compiler

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES

MAQAO feed back on dgemm



MAQAO feed back on dgemm



MAQAO feed back on dgemm

MAQAO

File Static Analysis Instrumentation Help

Project | dgemm_orig.s | produitMatriceNaive#

Zoom Init | Zoom In | Zoom Out

dgemm_orig.s | dgemm_orig.c | Sql | IPC | Oracle Report | Oracle Query

```
1        //#define SIZEL3 900
2        //#define SIZEL2 180
3        //#define SIZE_MATRIX 1800
4        #define SIZERX 8
5        //#define SIZECY 200
6        //#define SIZECX 200
7        #define SIZE 100
8        #define SIZEK 200
9        //#define SIZE 200
10
11
12
13       void produitMatriceNaive(double *A,double *B,double *C,int n)
14       {
15       int i,j,k,ii,jj;
16
17       for(i=0;i<n;i++) {
18        for(j=0;j<n;j++) {
19          for (k=0;k<n;k++){
20                    C[i*n+j]+=A[i*n+k]*B[k*n+j];
21
22                         }
23                       }
24                     }
25
26       }
```

Filter

Static

Loop id	Static estimated cycles	Issues/Bound ra
0	7*N	7
3	8*N + 16	1.6
4	8*N + 16	1.33

Ok

dgemm - Konqueror | MAQAO | topaz@a6bgl6: ~/lsty/maqa

06:12 2006-08-06

MAQAO feed back on dgemm

MAQAO

File Static Analysis Instrumentation Help

Project | dgemm_orig_mod.s | produitMatriceNaive#

Zoom Init | Zoom In | Zoom Out

dgemm_orig_mod.s | dgemm_orig_mod.c | Sql | IPC | Oracle Report | Oracle Query

```
1       //#define SIZEL3 900
2       //#define SIZEL2 180
3       //#define SIZE_MATRIX 1800
4       #define SIZERX 8
5       //#define SIZECY 200
6       //#define SIZECX 200
7       #define SIZE 100
8       #define SIZEK 200
9       //#define SIZE 200
10
11
12
13      void produitMatriceNaive(double **A,double **B,double **C,int n)
14      {
15      int i,j,k,ii,jj;
16
17      for(i=0;i<n;i++) {
18       for(j=0;j<n;j++) {
19         for (k=0;k<n;k++){
20             //C[i*n+j]+=A[i*n+k]*B[k*n+j];
21                  C[i][j]+=A[i][k]*B[k][j];
22
23                              }
24                            }
25                          }
26
27      }
```

images - Konqueror | MAQAO | My Way - Mozilla Firefox

otomaz@allaoua:~/dgemm | Session par défaut: file:///hc

06:24 2006-08-06

MAQAO feed back on dgemm

MAQAO

File   Static Analysis   Instrumentation   Help

Project | dgemm_orig_mod.s | produitMatriceNaive#

Zoom Init | Zoom In | Zoom Out

dgemm_orig_mod.s

```
1        //#define SIZ
2        //#define SIZ
3        //#define SIZ
4        #define SIZEF
5        //#define SIZ
6        //#define SIZ
7        #define SIZE
8        #define SIZEF
9        //#define SIZ
10
11
12
13       void produitM
14       {
15       int i,j,k,ii,
16
17       for(i=0;i<n;i
18        for(j=0;j<n;
19          for (k=0;k
20              //C[i*
21                  C
22
23
24
25
26
27       }
```

Results of static analysis

Loop ID : 2
Source Line : 21

Static stats for loop 2

Report
Issue/Optimal bound ranges greater than 3;Suspicious scheduling within loop body
You should unroll and bet on pipelining

Report
Check.s detected, possible code speculation
Speculative code may harms performance ; Check source code for while loop
You should turn off speculation

Report
Detected a loop version due to LOADS PAIR

Report
Loop part of a 3-ways versioned loop

Loop ID : 4
Source Line : 21

Static stats for loop 4

Report
Issue/Optimal bound ranges greater than 3;Suspicious scheduling within loop body
You should unroll and bet on pipelining

Report
Check.s detected, possible code speculation
Speculative code may harms performance ; Check source code for while loop
You should turn off speculation

Report
Detected a loop version due to LOADS PAIR

Report
Loop part of a 3-ways versioned loop

Ok

images - Konqueror | otomaz@allaoua:~/dgemm | MAQAO | Session par défaut: file:///hc | My Way - Mozilla Firefox | 06:25 2006-08-06

MAQAO feed back on dgemm

MAQAO

File Static Analysis Instrumentation Help

Project | dgemm_orig_mod.s | produitMatriceNaive#

Zoom Init | Zoom In | Zoom Out

dgemm_orig_mod.s | dgemm_orig_mod.c | Sql | IPC | Oracle Report | Oracle Query

```
1        //#define SIZEL3 900
2        //#define SIZEL2 180
3        //#define SIZE_MATRIX 1800
4        #define SIZERX 8
5        //#define SIZECY 200
6        //#define SIZECX 200
7        #define SIZE 100
8        #define SIZEK 200
9        //#define SIZE 200
10
11
12
13       void produitMatriceNaive(double **A,double **B,double **C,int n)
14       {
15       int i,j,k,ii,jj;
16
17       for(i=0;i<n;i++) {
18        for(j=0;j<n;j++) {
19          for (k=0;k<n;k++){
20               //C[i*n+j]+=A[i*n+k]*B[k*n+j];
21                    C[i][j]+=A[i][k]*B[k][j];
22
23                                   }
24                                 }
25                               }
26
27       }
```

Filter

Static

Loop id	Static estimated cycles	Issues/Bound ra
4	164*N	16.4
2	37*N	9.25
6	5*N	5

Ok

images - Konqueror | otomaz@allaoua:~/dgemm | MAQAO | Session par défaut: file:///hc | My Way - Mozilla Firefox | 2 | 06:25 2006-08-06

Conclusion

- Search is necessary due to complex architecture
- Possible to limit this search to lower level of optimizations and use of performance tuning tools
- Model necessary to eliminate combinatorial explosion for other transformations

UNIVERSITÉ DE VERSAILLES SAINT-QUENTIN-EN-YVELINES