

---

# Automatic Benchmarking and Optimization of codes: An Experience with Numerical Kernels

---

José R. Herrero, Juan J. Navarro

{josepr,juanjo}@ac.upc.es

Dept. Arquitectura Computadors  
Universitat Politècnica de Catalunya



# Talk Outline

---

- Automatic Benchmarking
- Optimization of (a type of) Numerical Algorithms

# Overview: Optimization of codes

- Multiple variants of code
- Multiple input parameters
- Multiple target platforms
  - Different optimum

⇒ Large number of combinations

⇒ Need to automate optimization process

# The need for automatization

---

- Information about ALL input variables must be kept
  - Compile time variables
    - Preprocessor symbols
    - Compiler flags
  - Execution time variables
    - Command line flags
    - environment variables

Otherwise results can become useless

# Automatic BenchMarking Tool

Launches

Compilations

Executions

Controlled  
by

Compile time vars  
•Preprocessor symbols  
•Compiler flags

Exec time vars  
•Binary  
•options

Keeps

Binary file

Results file

Named

Benchmark\_name

Binary\_name

+

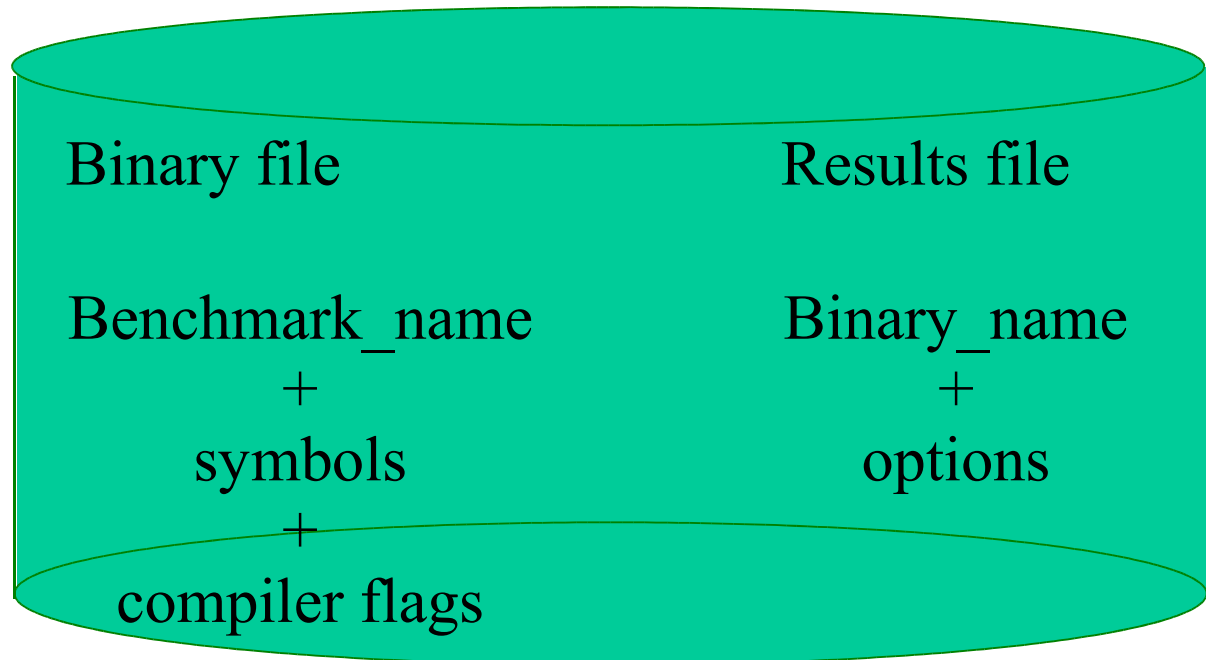
symbols

+

options

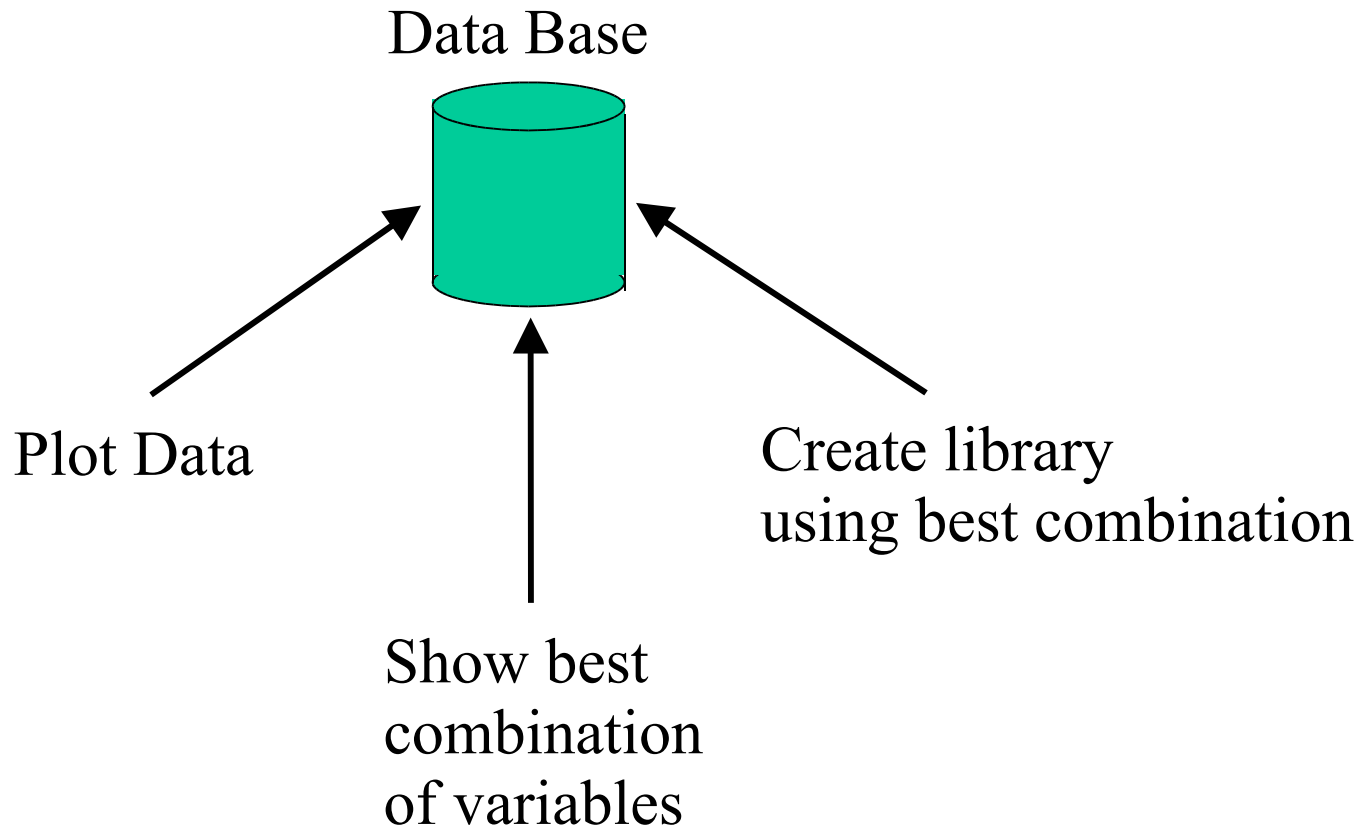
+

compiler flags



# BMT: overview

---



# Benchmarking: Important aspects

- Handling a variable # of parameters
  - Simulate a variable number of control loops
    - Get all combinations
  
- Handling control variables of different types
  - Numbers, strings
    - We used Perl!
      - Symbolic References

# Benchmarking: Important aspects

- Concurrency
  - Use locks in DB
  - Use different directories for compilation
    - Maker (June 26th, 12:00 - G-SERP)
  - Use different filenames (both for binaries and executions results)

# Benchmarking: Important aspects

- Enforce correctness and robustness
  - Check if parameters were those expected
  - Check if program terminated normally

# Application: an example

---

- Some applications perform many operations on very small matrices (ej.  $8 \times 8$ )
  - Multimedia codes
    - 3D graphics
    - Digital Signal Processing
    - ...
  - Sparse matrix codes
    - Hypermatrices

# Related work

---

- Efficient routines
  - Multimedia vector extensions
    - Modern, efficient, ...
    - But ...
      - There are no good compilers as yet
      - Machine languages are different
  - BLAS
    - Optimized to work efficiently on large matrices

# Goal

---

- Get routines which
    - Work on small matrices (which fit in cache)
    - Are efficient
    - Can be created for any target system
- ⇒ Create a Small Matrix Library (SML)

# Idea

---

- Fix parameters at compilation time
  - Leading dimensions
  - Loop limits
- Example:  $C=A*B^T$

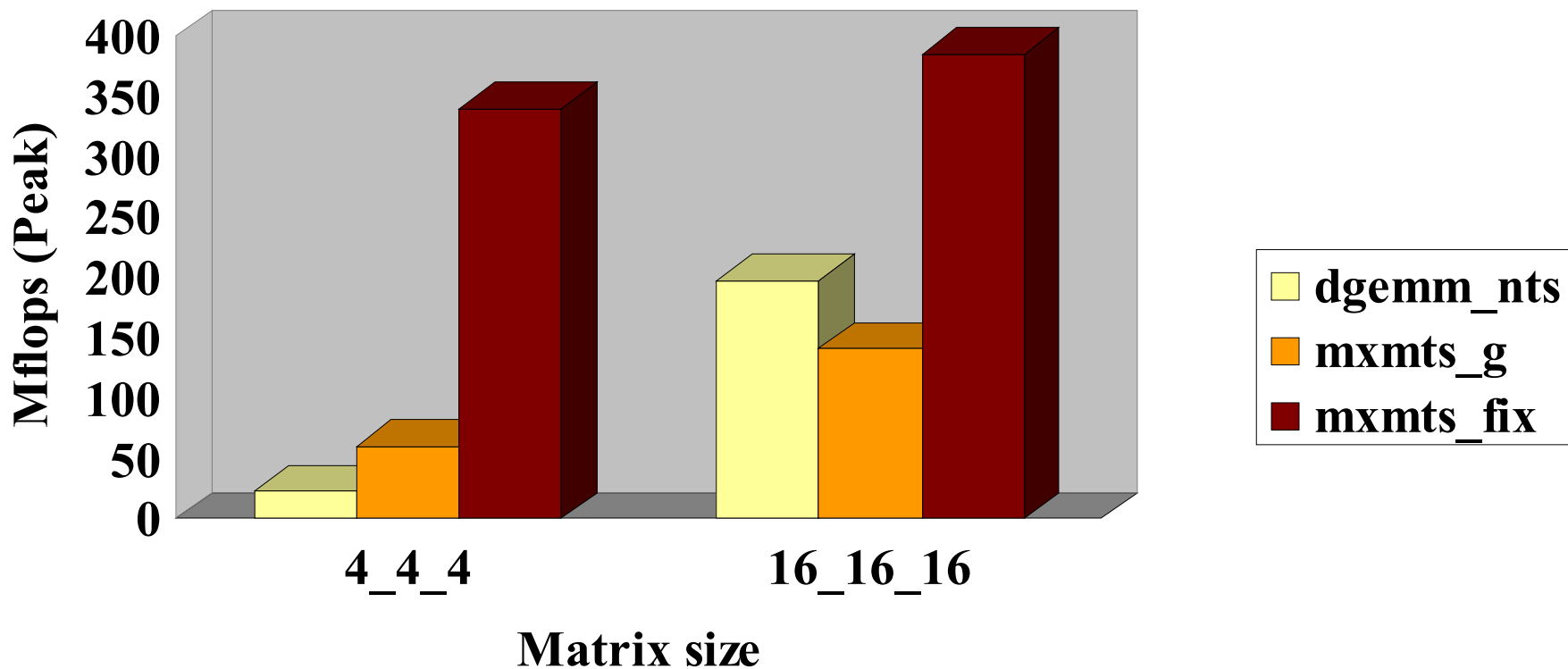
```
subroutine mxmt(A,B,C,  
  lda,ldb,ldc, ui, uj, uk)  
integer A(lda,*), ...  
do I=1, ui  
  ...
```



```
subroutine mxmt_fix(A,B,C)  
  
integer A(8,*)  
do I=1, 8  
  ...
```

# Motivation

## Alpha 21164: Peak performance of routines tested



# Procedure

---

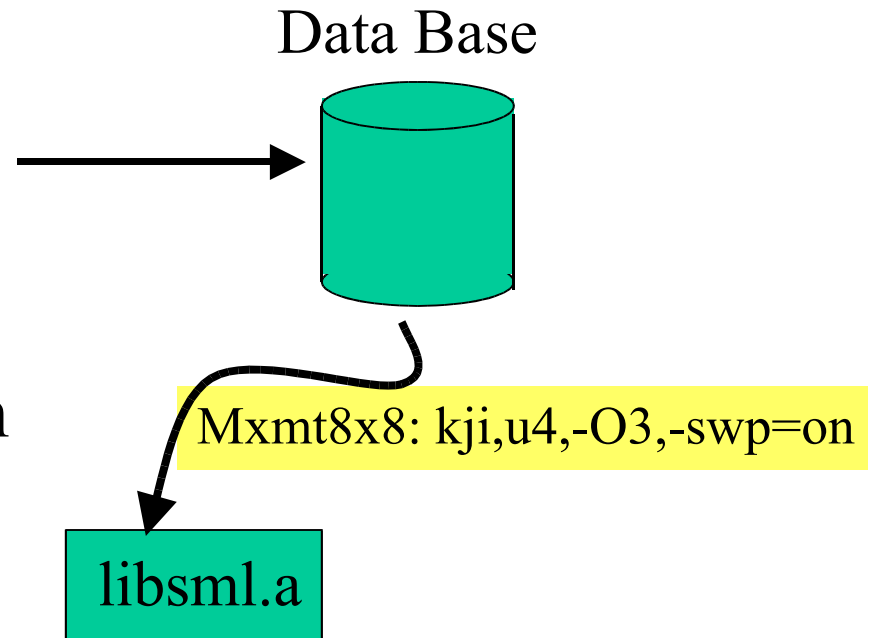
- Write several variants of code
  - Loop order
  - Loop unrolling
- Use the *best* compiler available
  - Try several compilation options
- Need to automate the tests!
  - Benchmarking tool

# Example: $M \times M^t$ codes

Code of Algorithm	form
1	$jik\_Creg$
2	$jik$
3	$kji\_Breg$
4	$i(jk4i\_BCreg)$
5	$i(jk4i\_Hreg)$
6	$i(jk4i)$
7	$ijk$
8	$kji$
9	$jki$
10	$jik4k\_Creg$
11	$jik8k\_Creg$

# Benchmarking Tool

- foreach parameter combination
  - ♦ compile
  - ♦ execute
  - ♦ store results (Mflops)
- Select best combination
- Add object to library

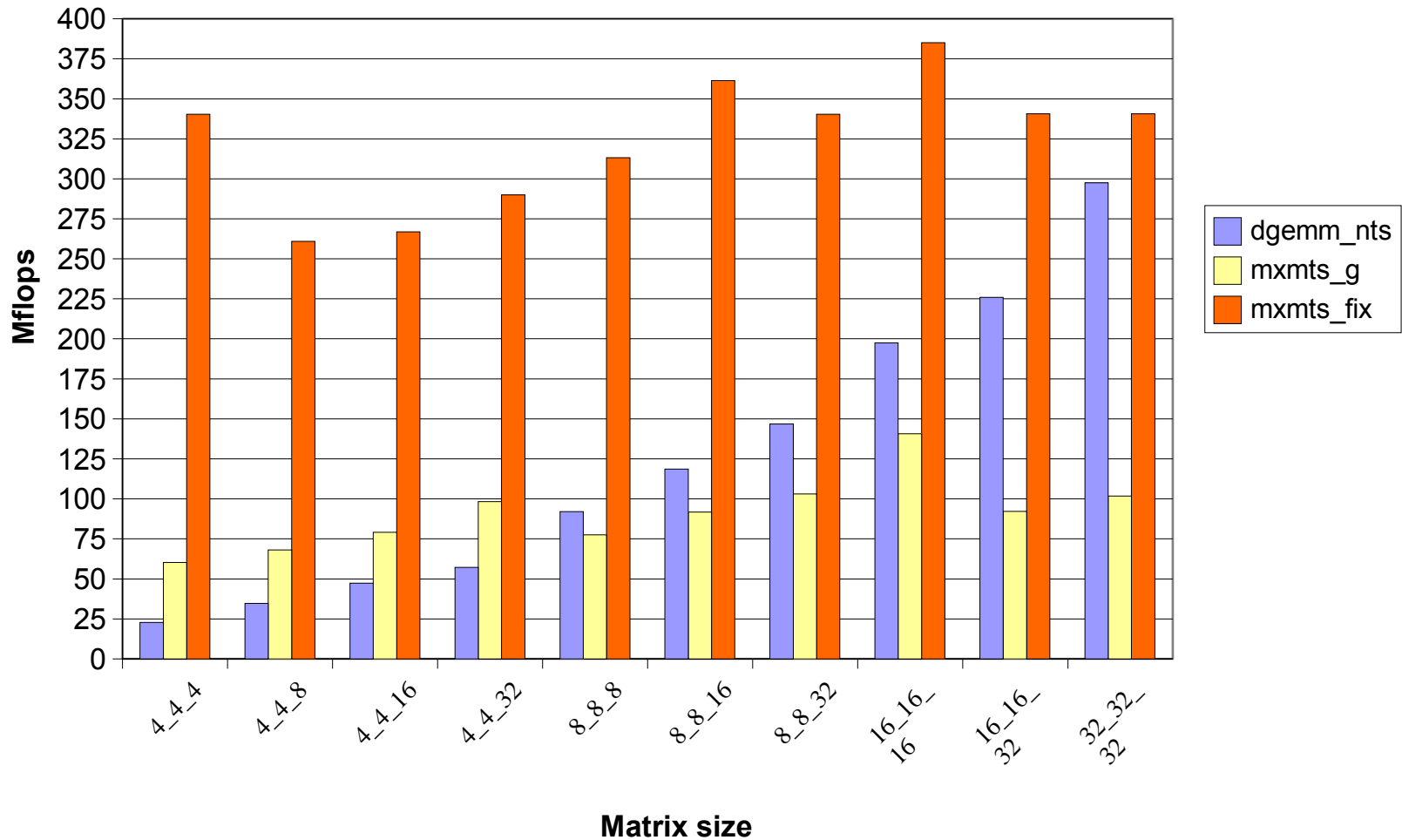


# Example: $M \times M^t$ optimum

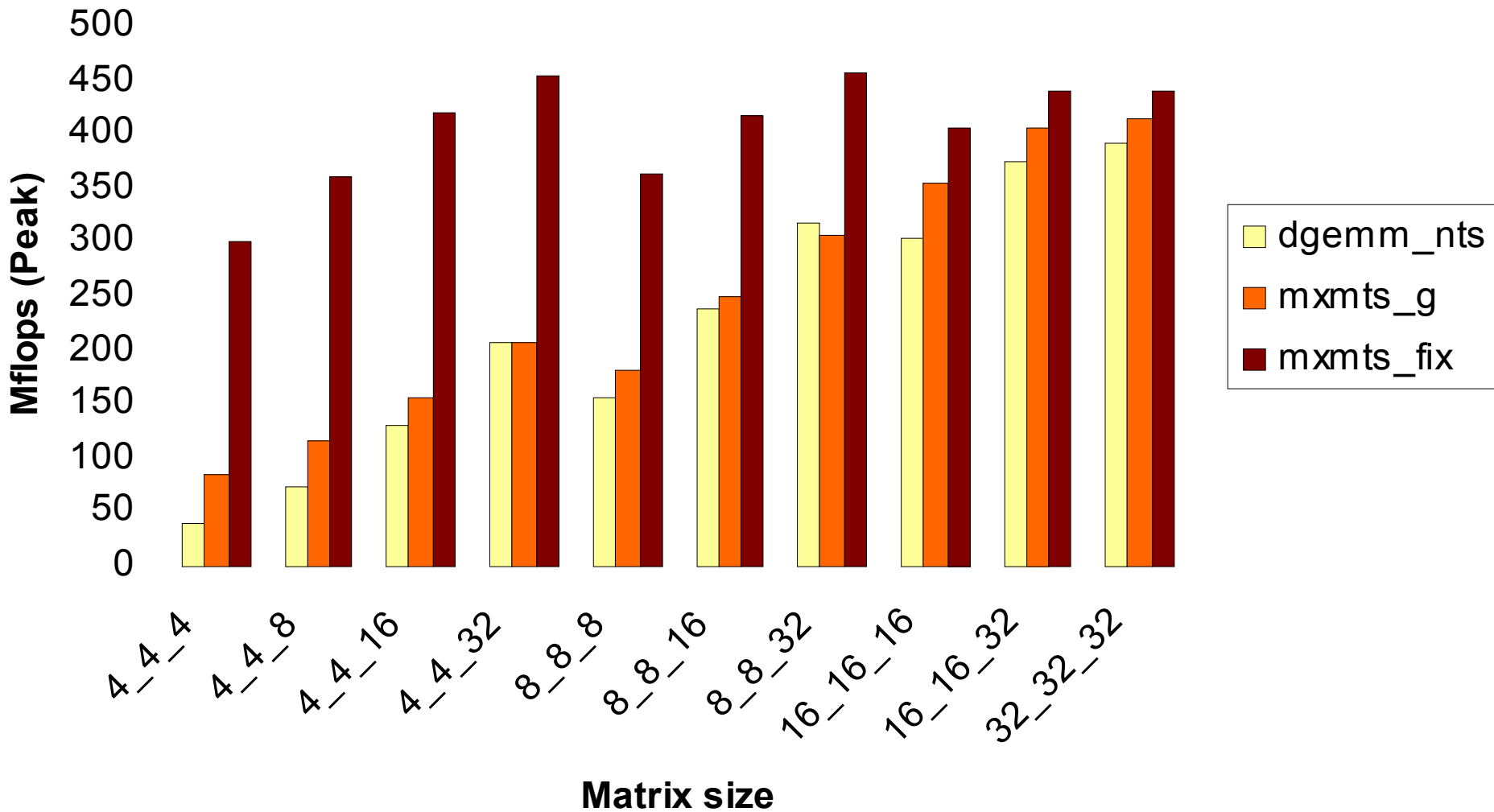
Code of Algorithm	form
1	$jik\_Creg$
2	$jik$
3	$kji\_Breg$
4	$i(jk4i\_BCreg)$
5	$i(jk4i\_Breg)$
6	$i(jk4i)$
7	$ijk$
8	$kji$
9	$jki$
10	$jik4k\_Creg$
11	$jik8k\_Creg$

Matrix sizes	<i>Alpha</i> 21164	<i>R10000</i>
4_4_4	2	8
4_4_8	3	5
4_4_16	3	3
4_4_32	3	3
8_8_8	10	6
8_8_16	11	8
8_8_32	11	5
16_16_16	11	2
16_16_32	11	5
32_32_32	11	6

# Results: Alpha-21164

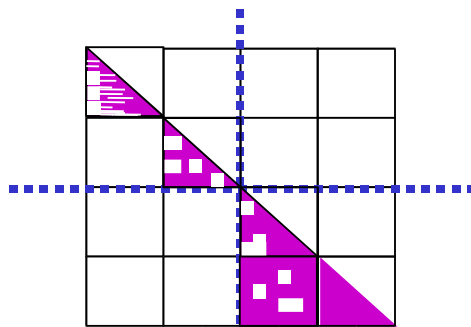


# R10000: Peak performance of routines tested

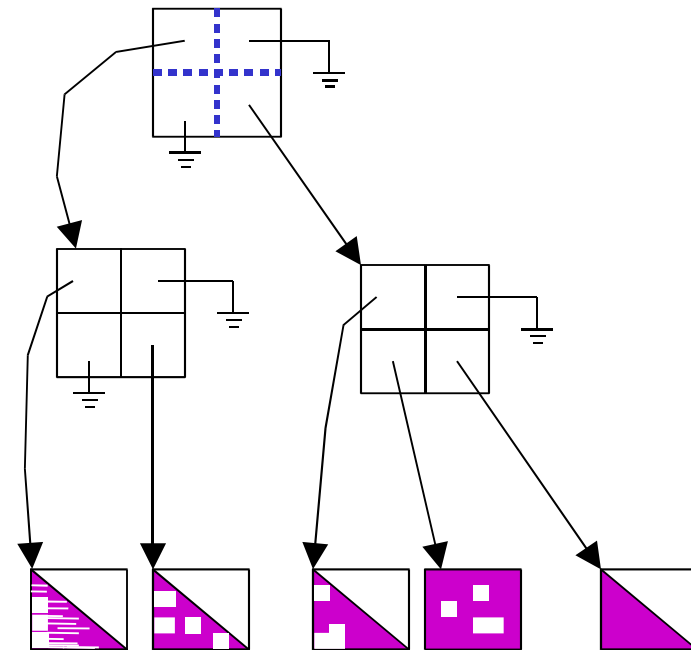


# Application

- Cholesky Factorization using Hypermatrices



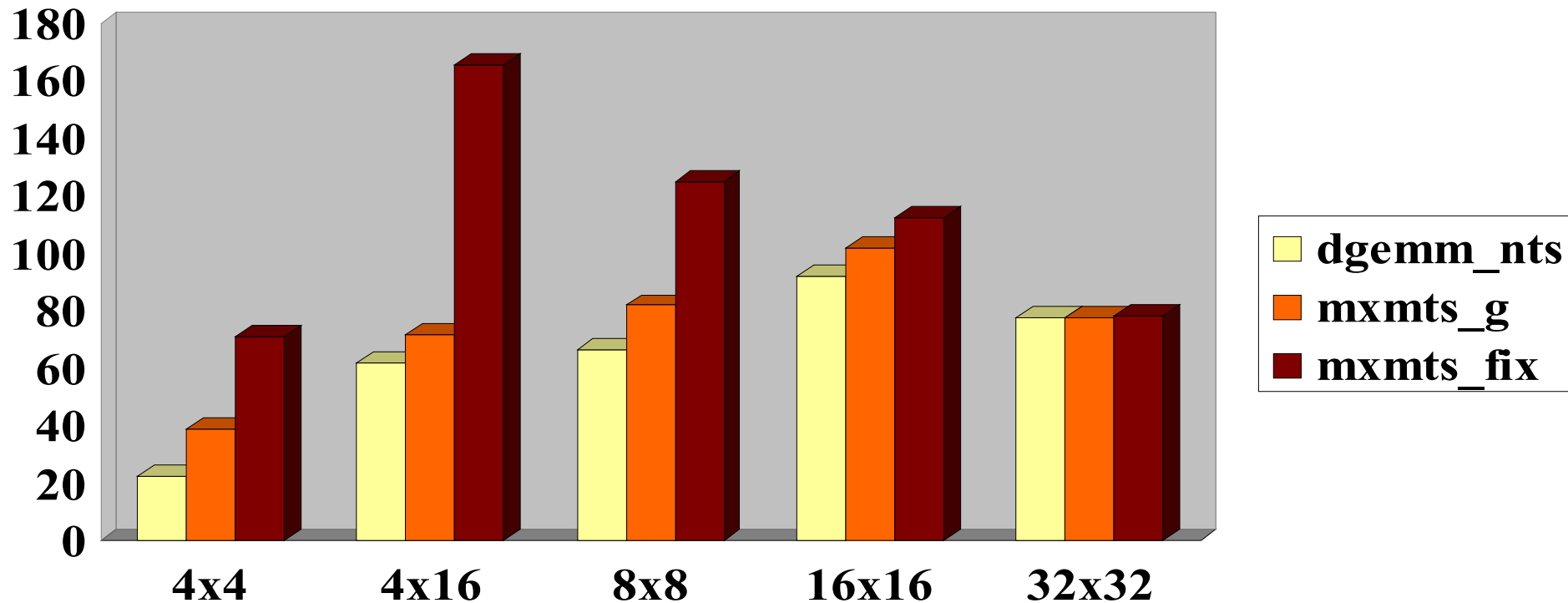
Matrix



HyperMatrix

# HM Cholesky

## R10000: Matriz bcsstk16



# Conclusions

---

- Keeping track of all information used when building and executing programs is
  - absolutely necessary and, at the same time,
  - difficult to manage by hand

⇒ We need to do it automatically
- Creation of ad-hoc codes can yield great performance but have a great creation cost