

# Reducing Overhead in Sparse Hypermatrix Cholesky Factorization

Josep R. Herrero, Juan J. Navarro  
{josepr,juanjo}@ac.upc.es

Computer Architecture Department  
Universitat Politècnica de Catalunya  
Barcelona  
Spain



# Outline

---

- Introduction
- Overhead reduction techniques
- Results
- Analysis
- Conclusions

# Introduction

---

Cholesky factorization

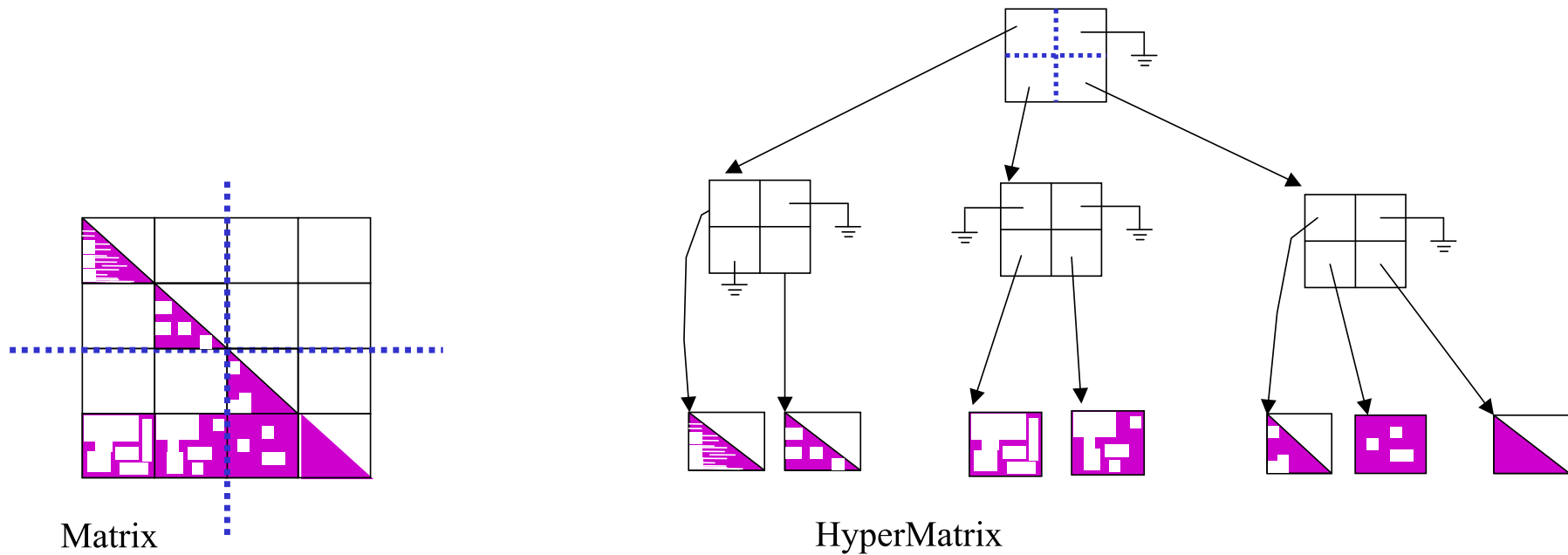
- Symmetric Positive Definite (SPD) matrix

Sparse matrices

- Plenty of 0's
- Avoid storage and computation on 0's

# Hypermatrix structure

---



# Overhead

---

Can store 0's within data submatrices

- Storage
- Computation

Trade-off in data submatrix size

- BLAS3 efficiency
- (Useless) operation on 0's

# Reducing Overhead & Increasing Performance

---

- Efficient kernels which operate on small data submatrices
- Bit Vectors associated to data submatrices
- Windows within data submatrices

# Efficient operation on small data submatrices

---

Goal:

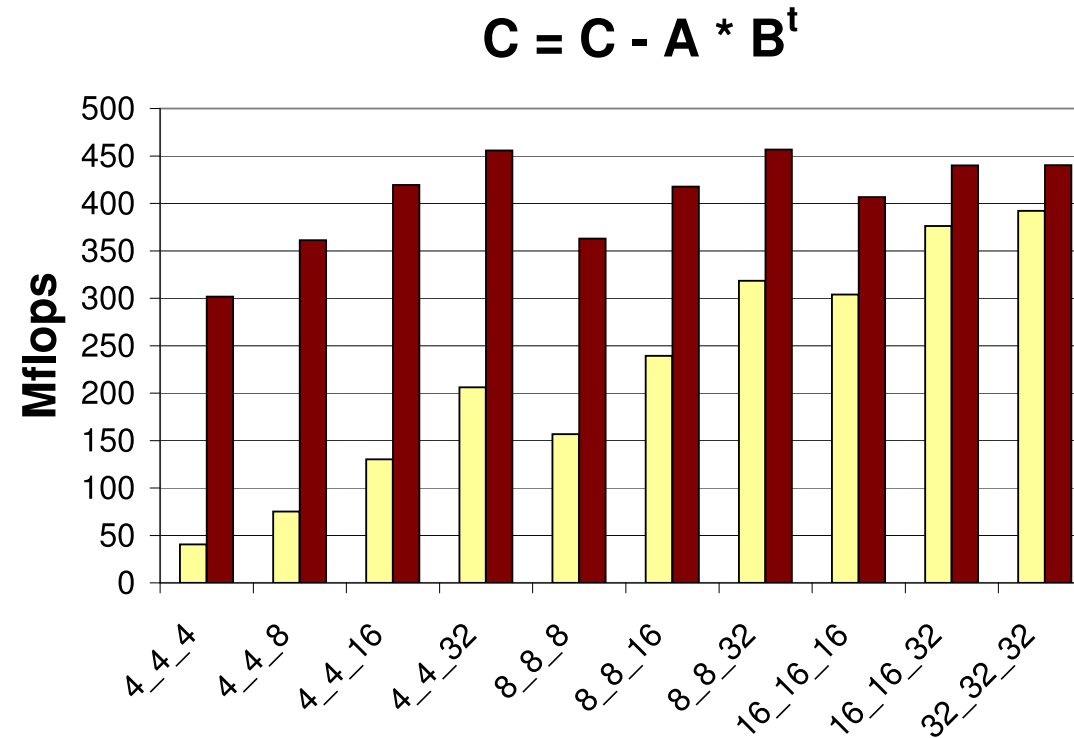
- Reduce data submatrix size while keeping good BLAS3 performance

Idea [Euro-Par'03]:

- Fix parameters at compilation time
- Choose best algorithm
  - Loop unrolling factors
  - Loop orders

# Matrix multiplication performance on small matrices

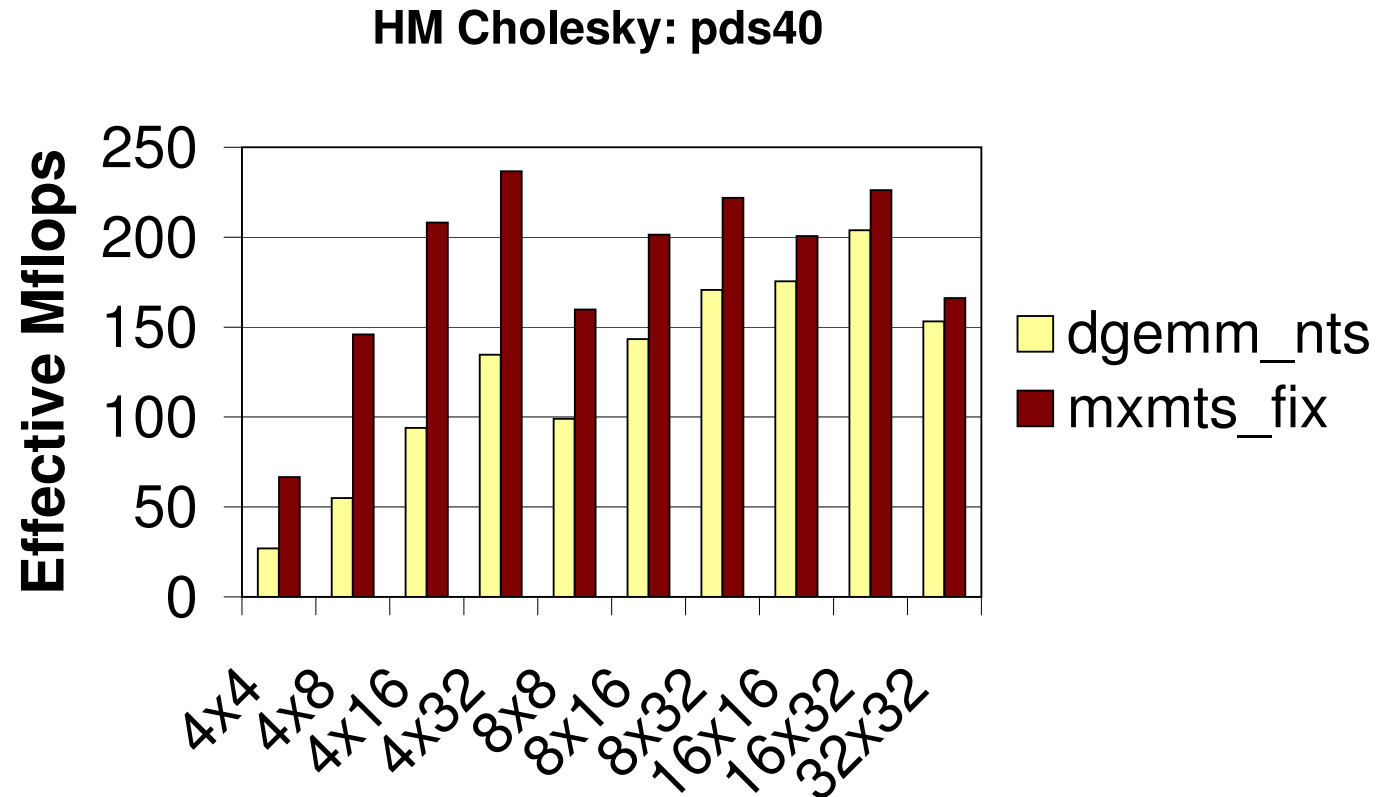
---



R10000 250 MHz (500 Mflops peak)

# Hypermatrix Cholesky on problem PDS40

---



LP problem: Patient Distribution System (40 days)

# Reducing Overhead & Increasing Performance

---

Operation on small data submatrices ...  
still has overhead

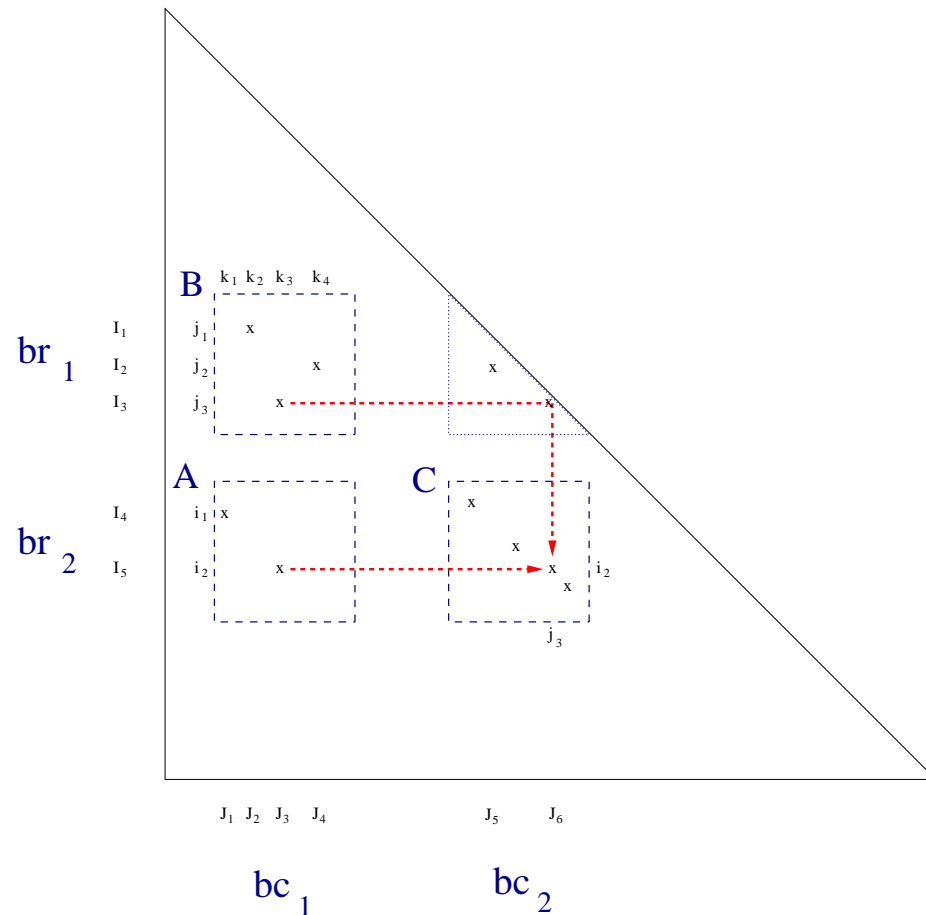
Goal: reduce the overhead further

- Bit Vectors associated to data submatrices
- Windows within data submatrices

# Bit Vectors: Goal

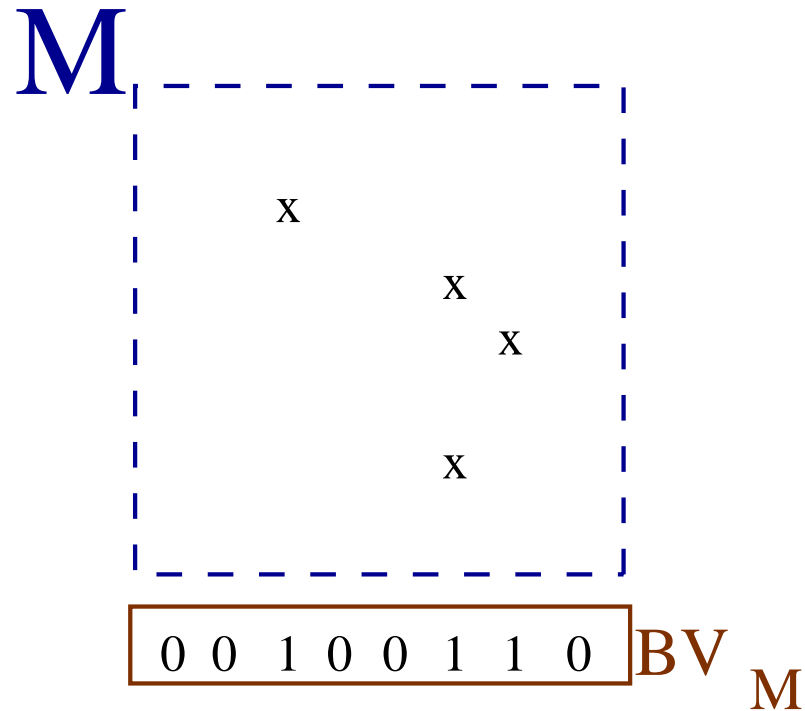
Reduce unnecessary computation

- Avoid matrix multiplication when two submatrices produce no update upon a third one



# Bit Vectors: Definition

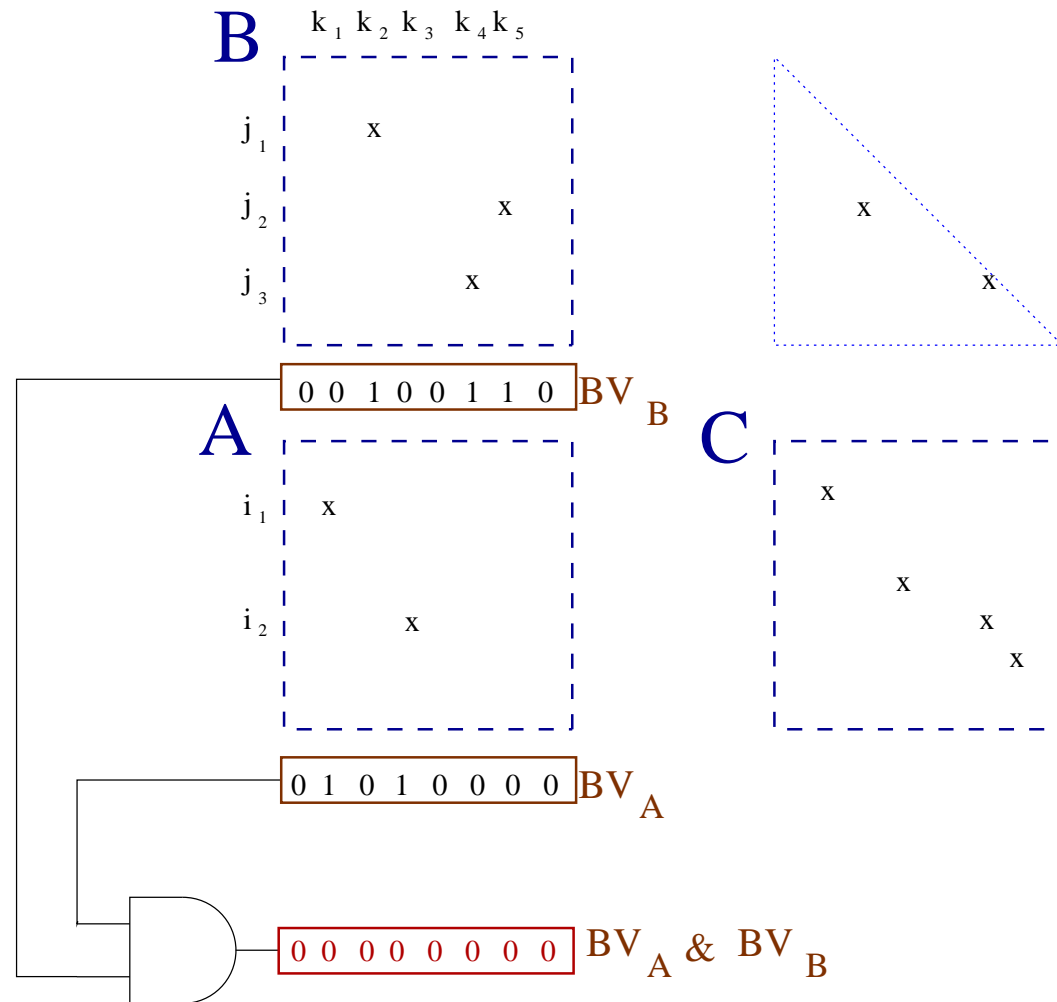
---



One bit associated to a column in a data submatrix

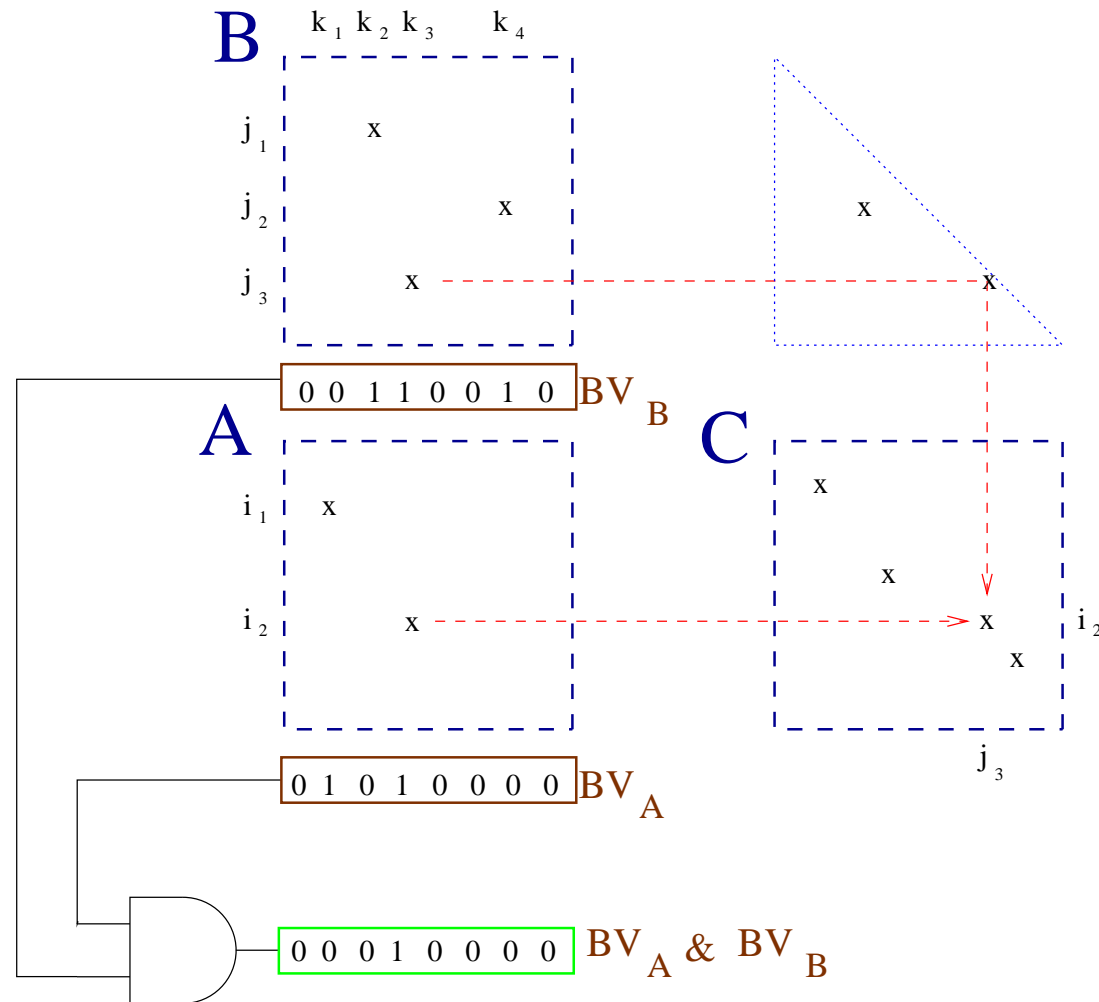
- Value = 0  $\Leftrightarrow$  column is full of 0's
- Value = 1  $\Leftrightarrow \exists \geq 1$  NZ in column

# Bit Vectors: Usage (I)



$BV_A \& BV_B = 0$ : Operation can be skipped

# Bit Vectors: Usage (II)



$BV_A \& BV_B \neq 0$ : Operation must be performed

# Windows within data submatrices: Goal

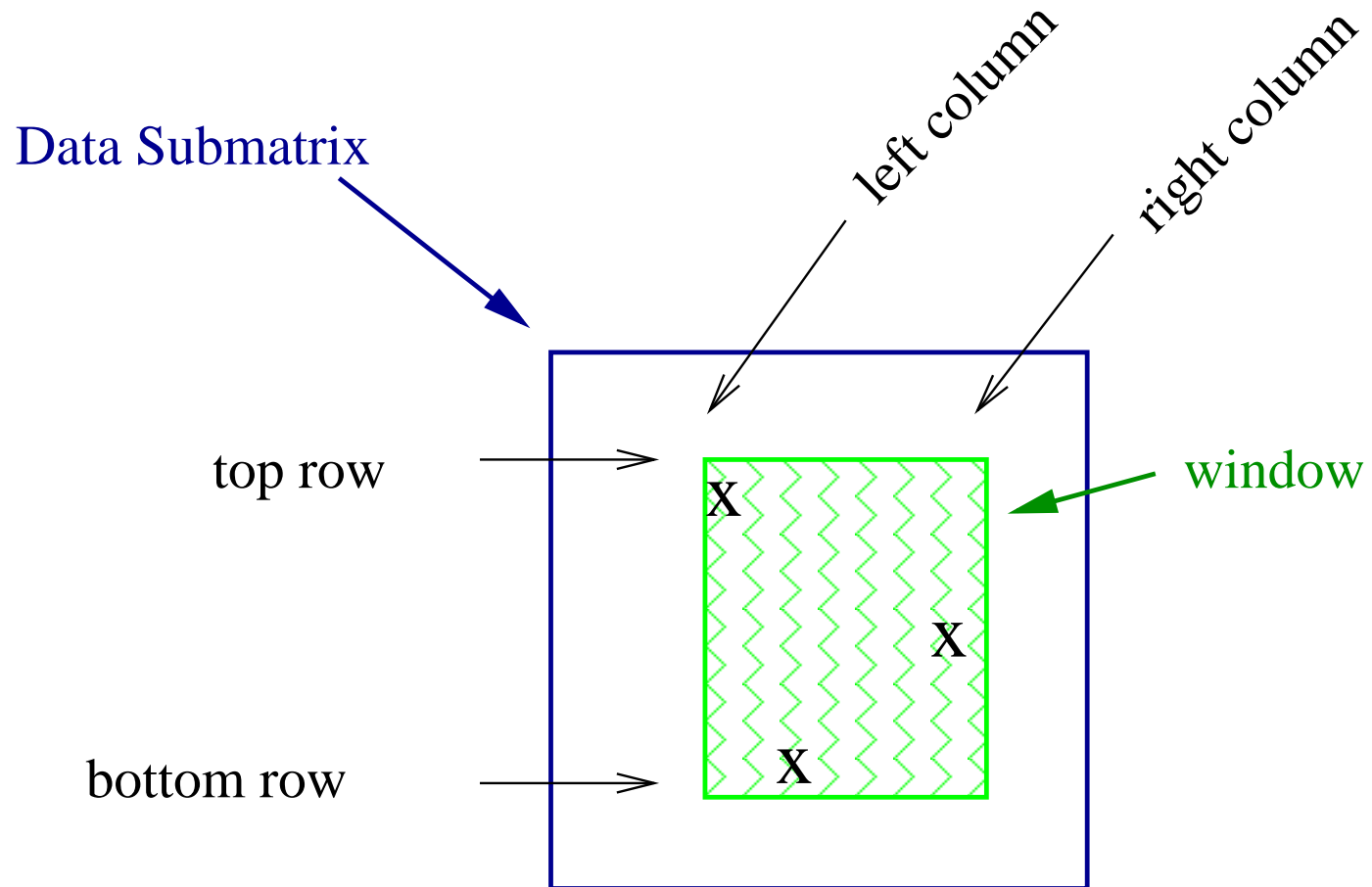
---

Store and use only a part of a data submatrix

- Reduce unnecessary computation
- Reduce storage

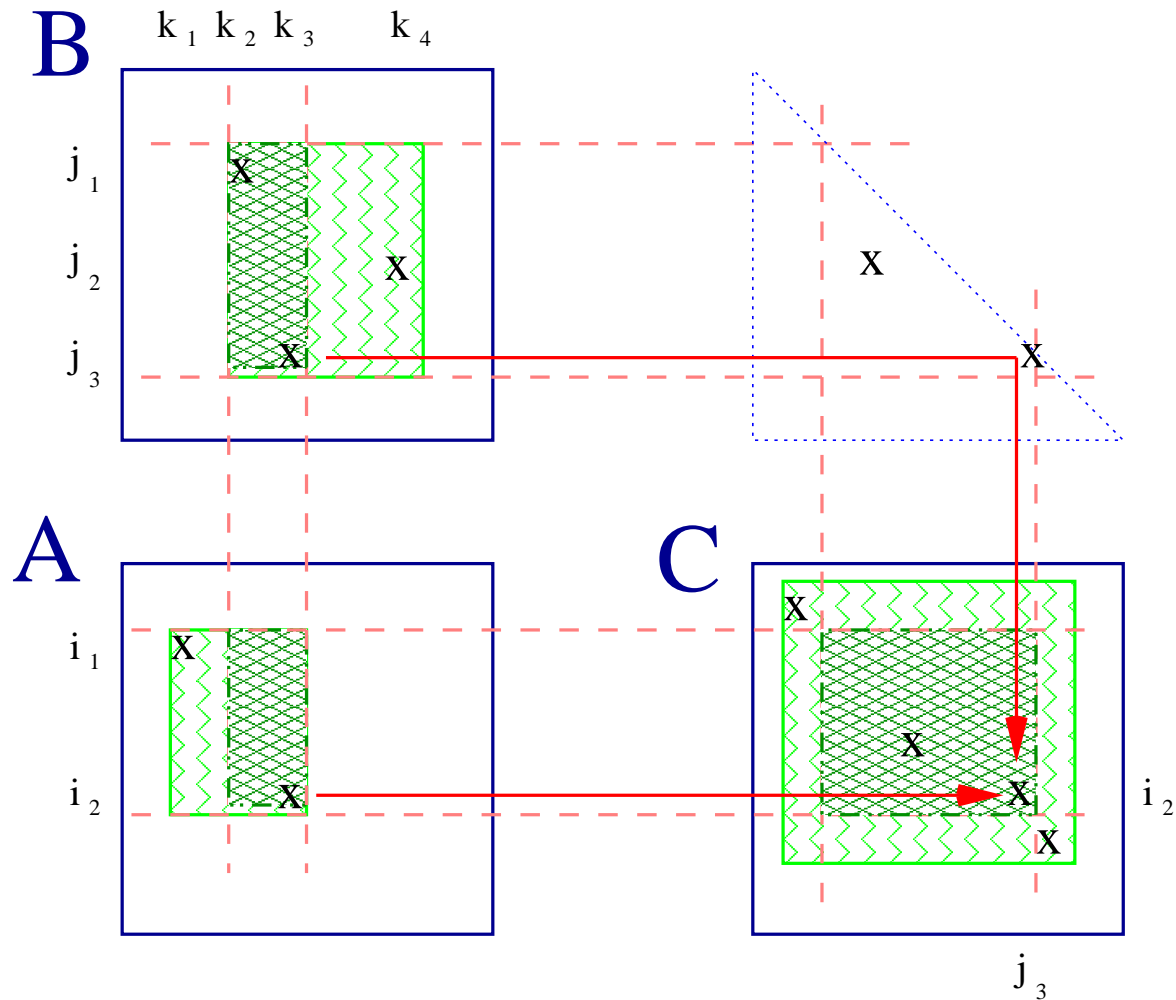
# Windows: Definition

---



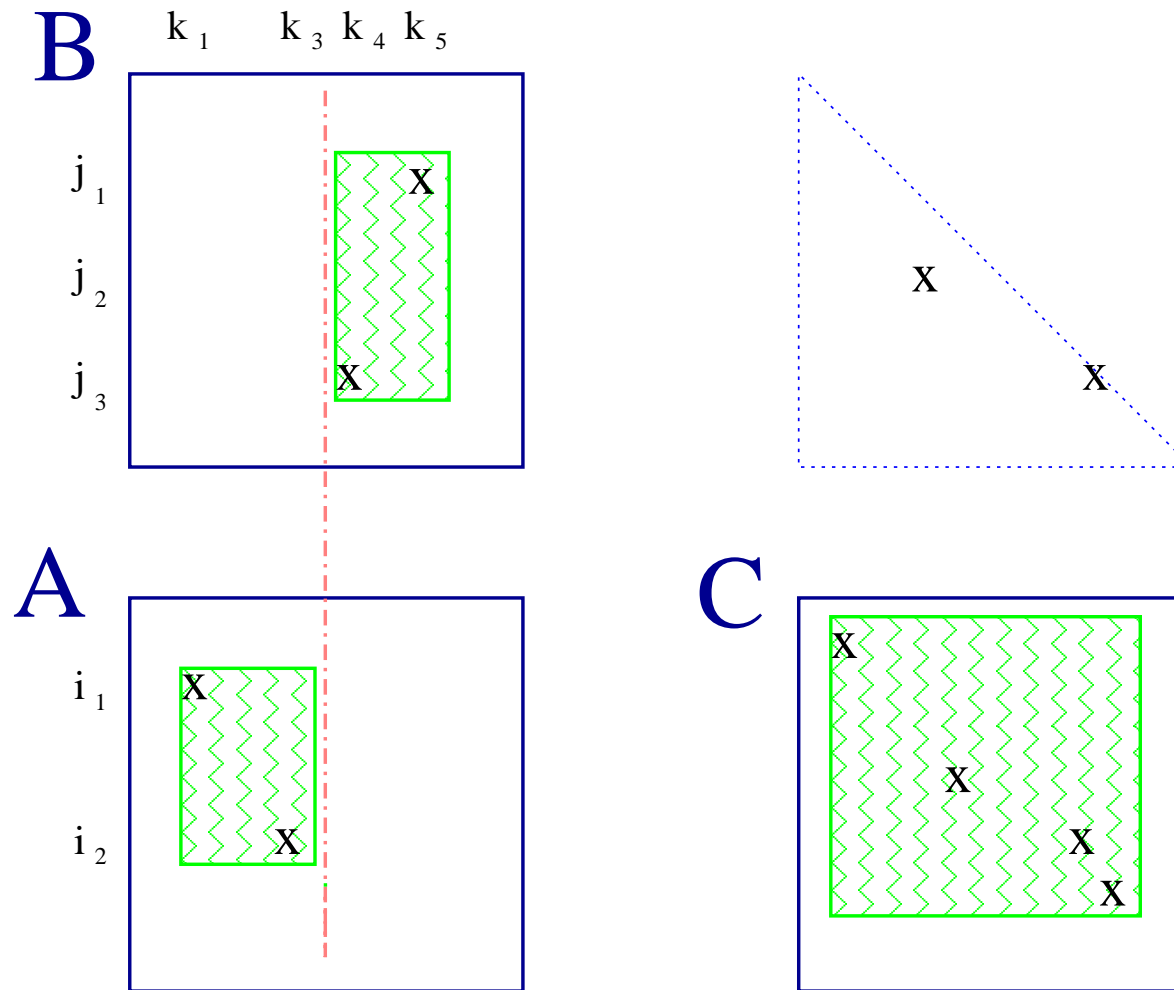
Window: subset of data submatrix

# Windows: Usage (I)



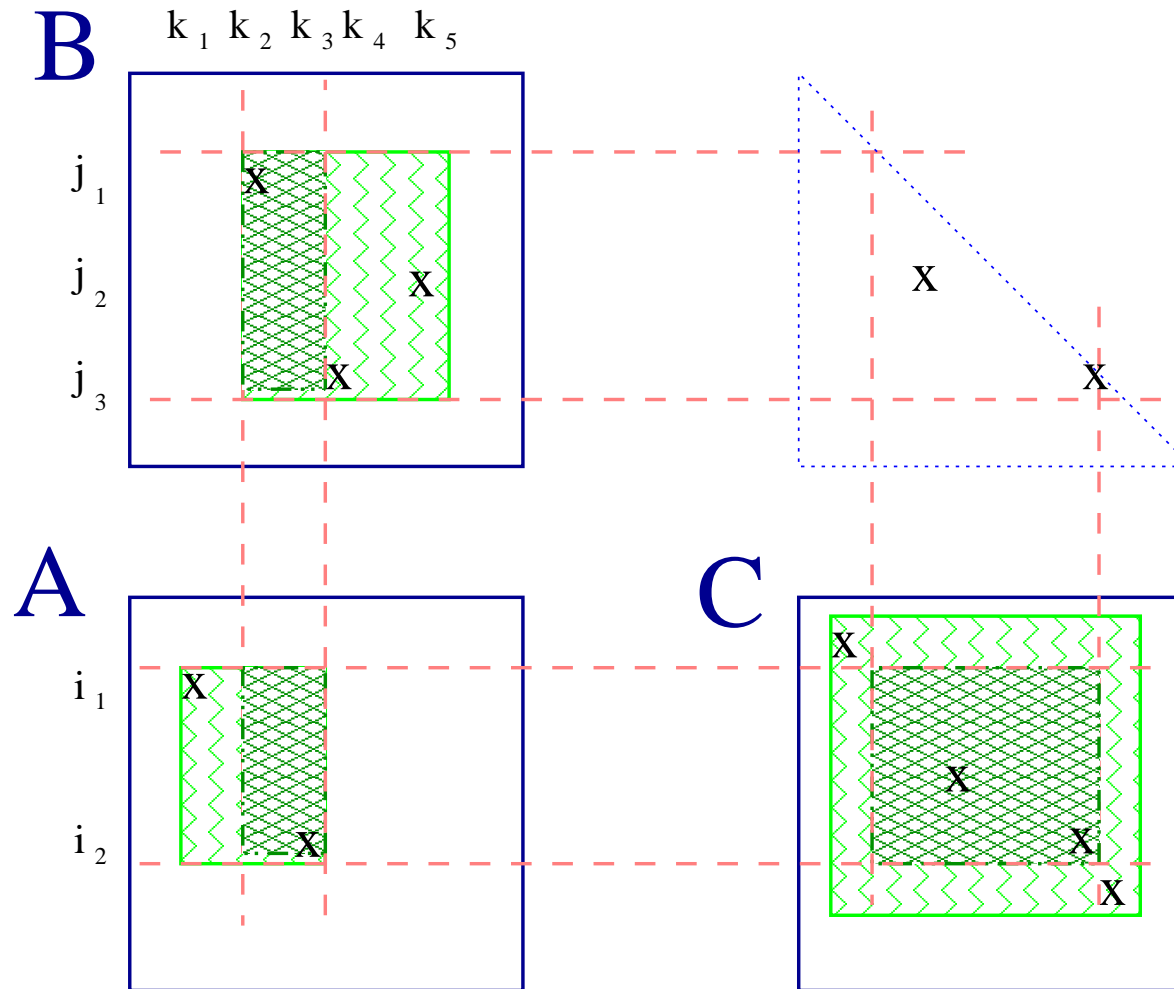
Operation can be reduced

# Windows: Usage (II)



Operation can be skipped

# Windows: Usage (III)



Unnecessary operation performed (Could be avoided with BVs)

# Results: Context information

---

- MIPS R10000 @ 250 MHz (500 Mflops peak)
- Sequential code
- Data submatrices of fixed size
- Large problems solved In-Core
- Ordered using METIS
- Linear Programming problems
  - NetLib
  - Multicommodity Network Flow generators

# Matrix characteristics

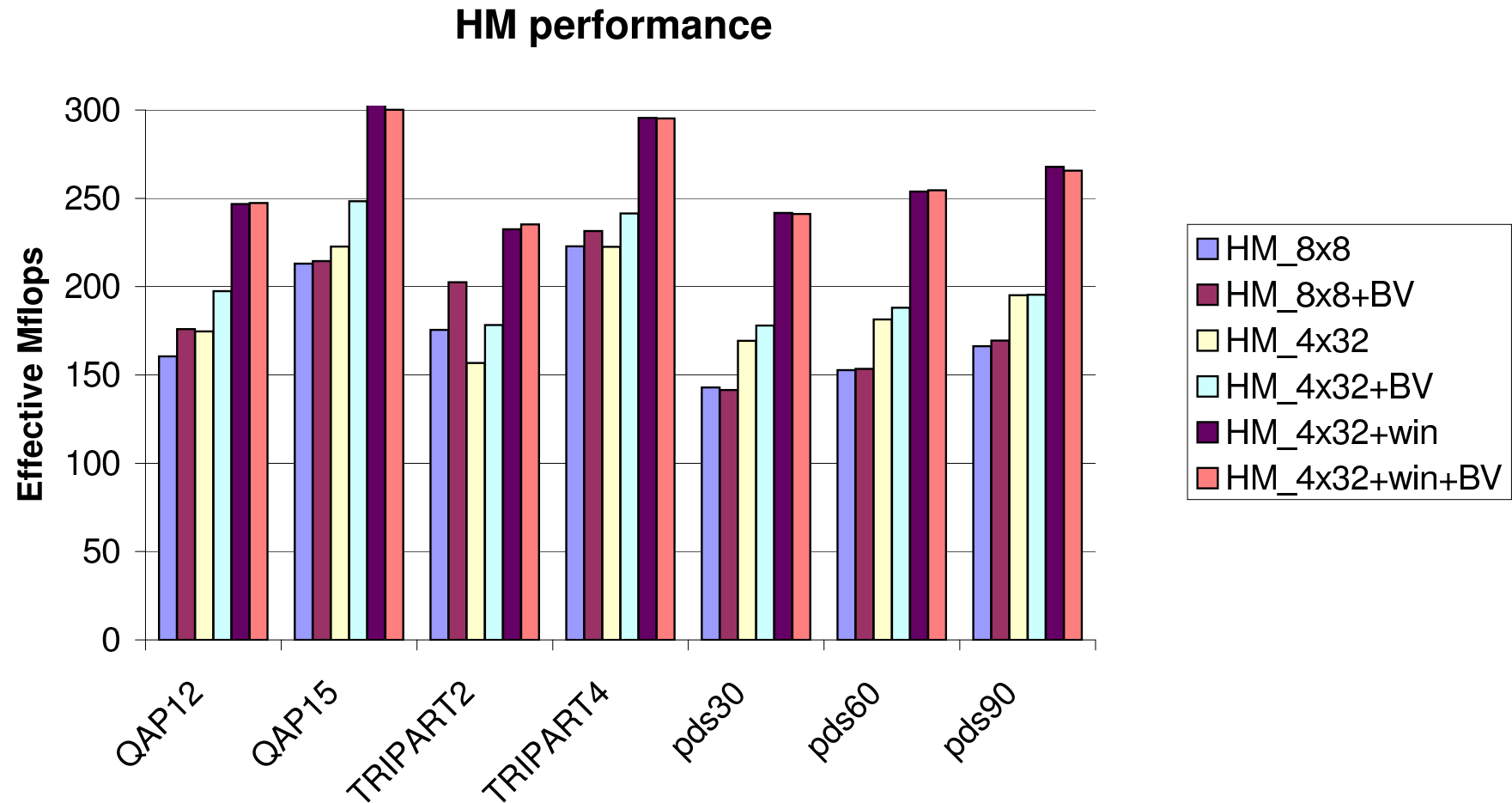
---

Matrix	Dimension	NZs	NZs in L <sup>a</sup>	Density	Flops to factor <sup>b</sup>
GRIDGEN1	330430	3162757	130586943	0.002	278891
QAP8	912	14864	193228	0.463	63
QAP12	3192	77784	2091706	0.410	2228
QAP15	6330	192405	8755465	0.436	20454
RMFGEN1	28077	151557	6469394	0.016	6323
TRIPART1	4238	80846	1147857	0.127	511
TRIPART2	19781	400229	5917820	0.030	2926
TRIPART3	38881	973881	17806642	0.023	14058
TRIPART4	56869	2407504	76805463	0.047	187168
pds1	1561	12165	37339	0.030	1
pds10	18612	148038	3384640	0.019	2519
pds20	38726	319041	10739539	0.014	13128
pds30	57193	463732	18216426	0.011	26262
pds40	76771	629851	27672127	0.009	43807
pds50	95936	791087	36321636	0.007	61180
pds60	115312	956906	46377926	0.006	81447
pds70	133326	1100254	54795729	0.006	100023
pds80	149558	1216223	64148298	0.005	125002
pds90	164944	1320298	70140993	0.005	138765

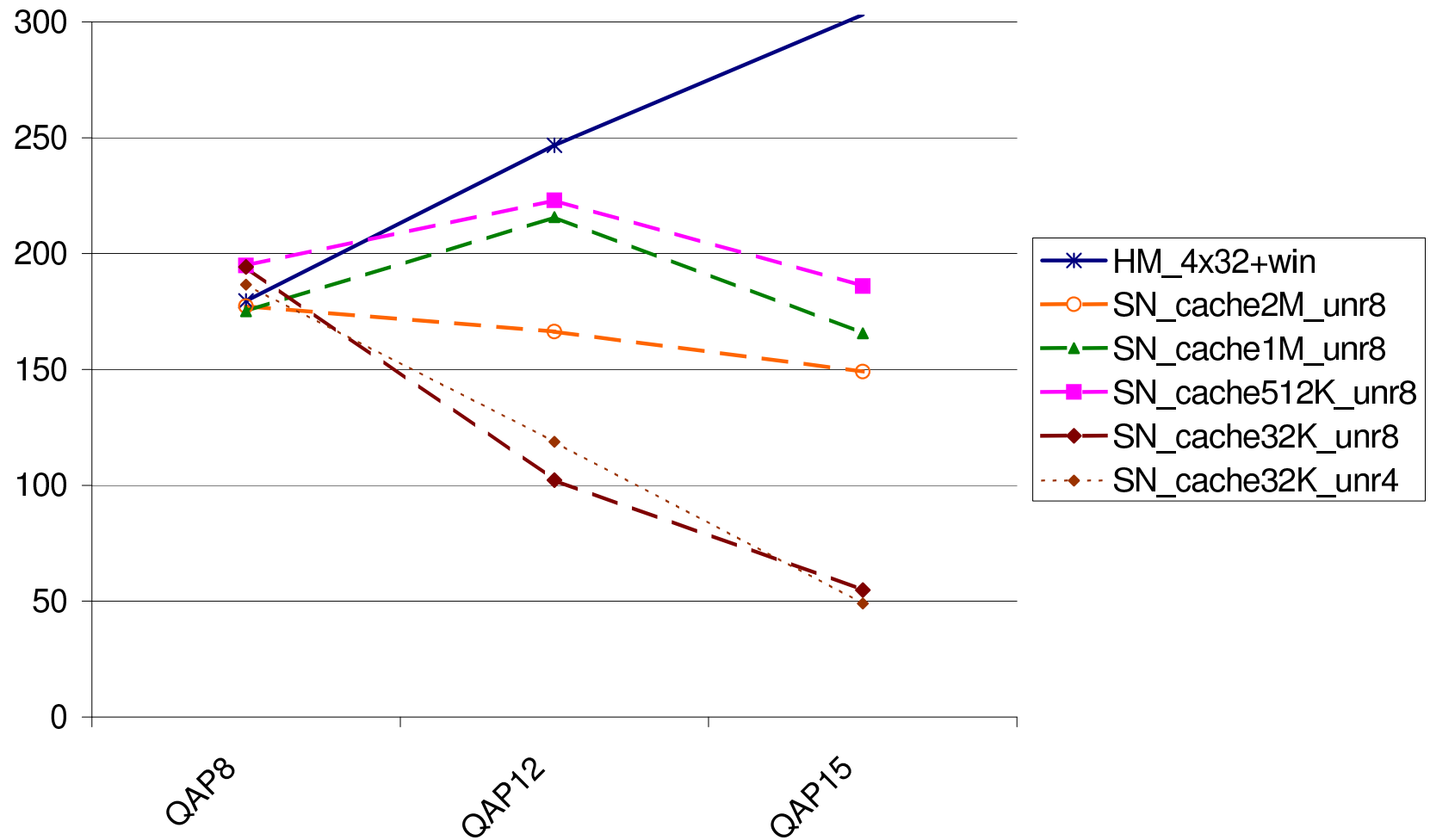
---

- <sup>a</sup>Number of non-zeros in factor L (matrix ordered using METIS).
- <sup>b</sup>Number of floating point operations (in Millions) necessary to obtain L from the original matrix (ordered with METIS).

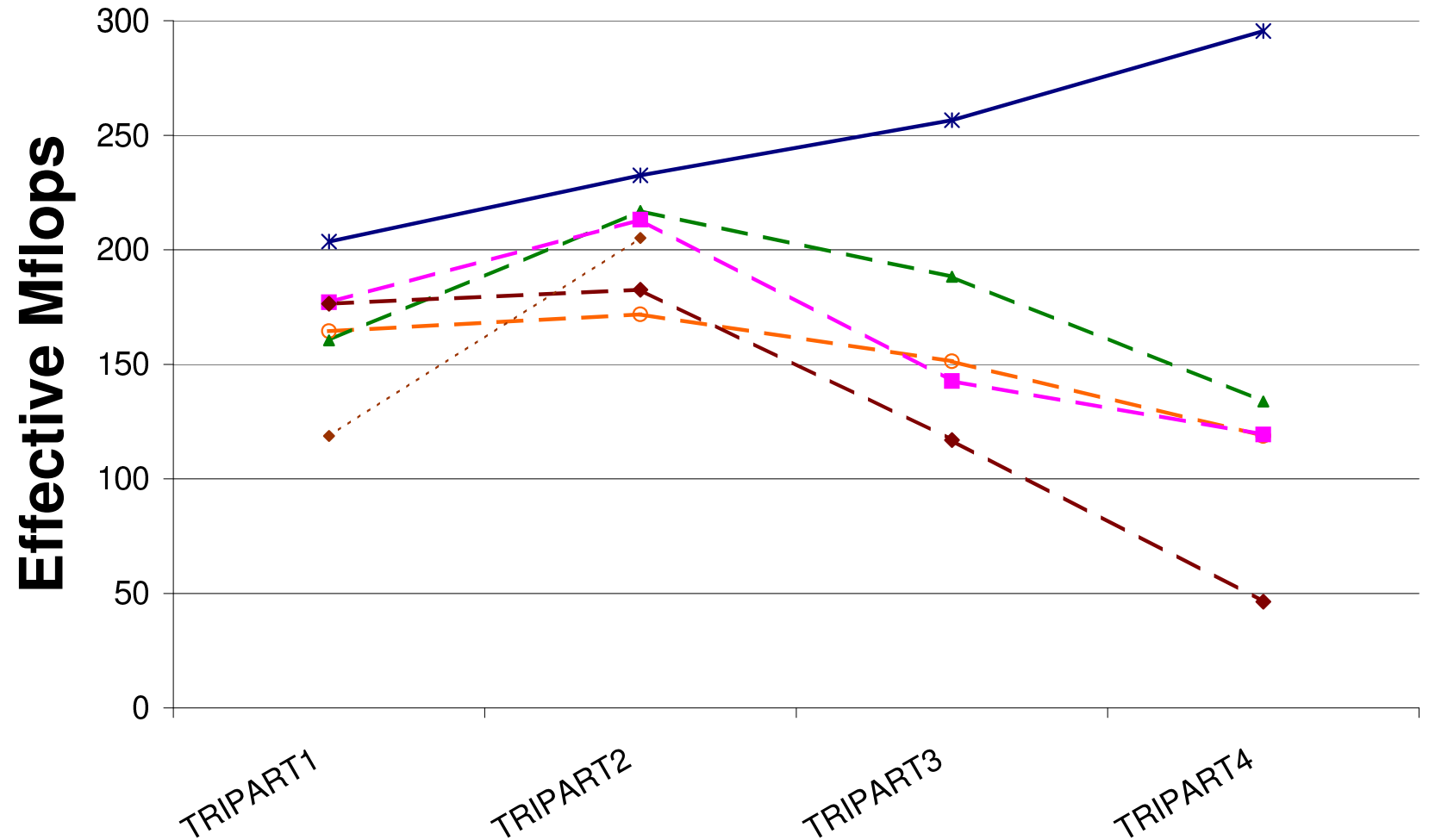
# Performance of several HM codes



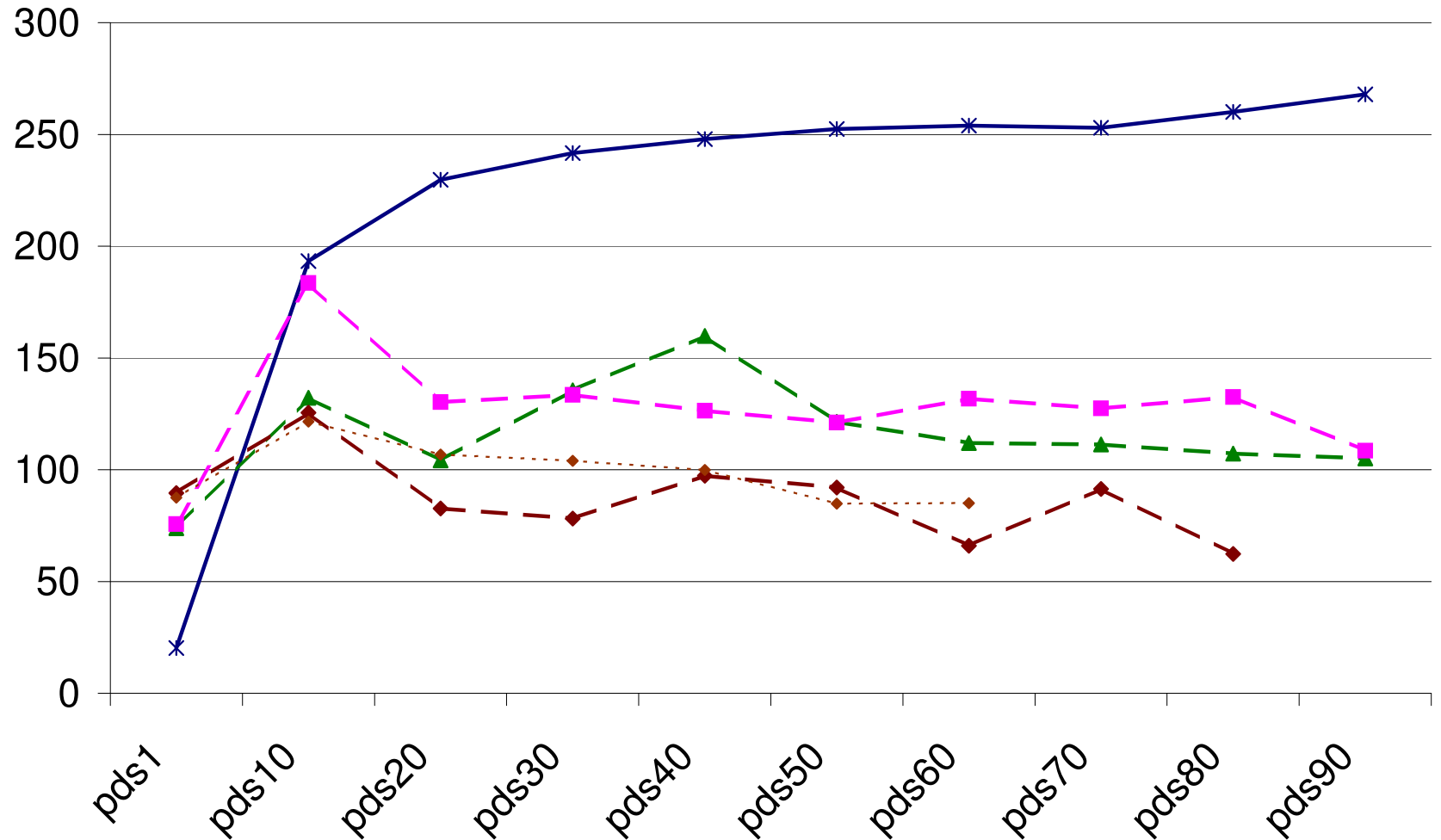
# HM vs SN (Ng-Peyton): QAP matrix family



# HM vs SN: TRIPART matrix family

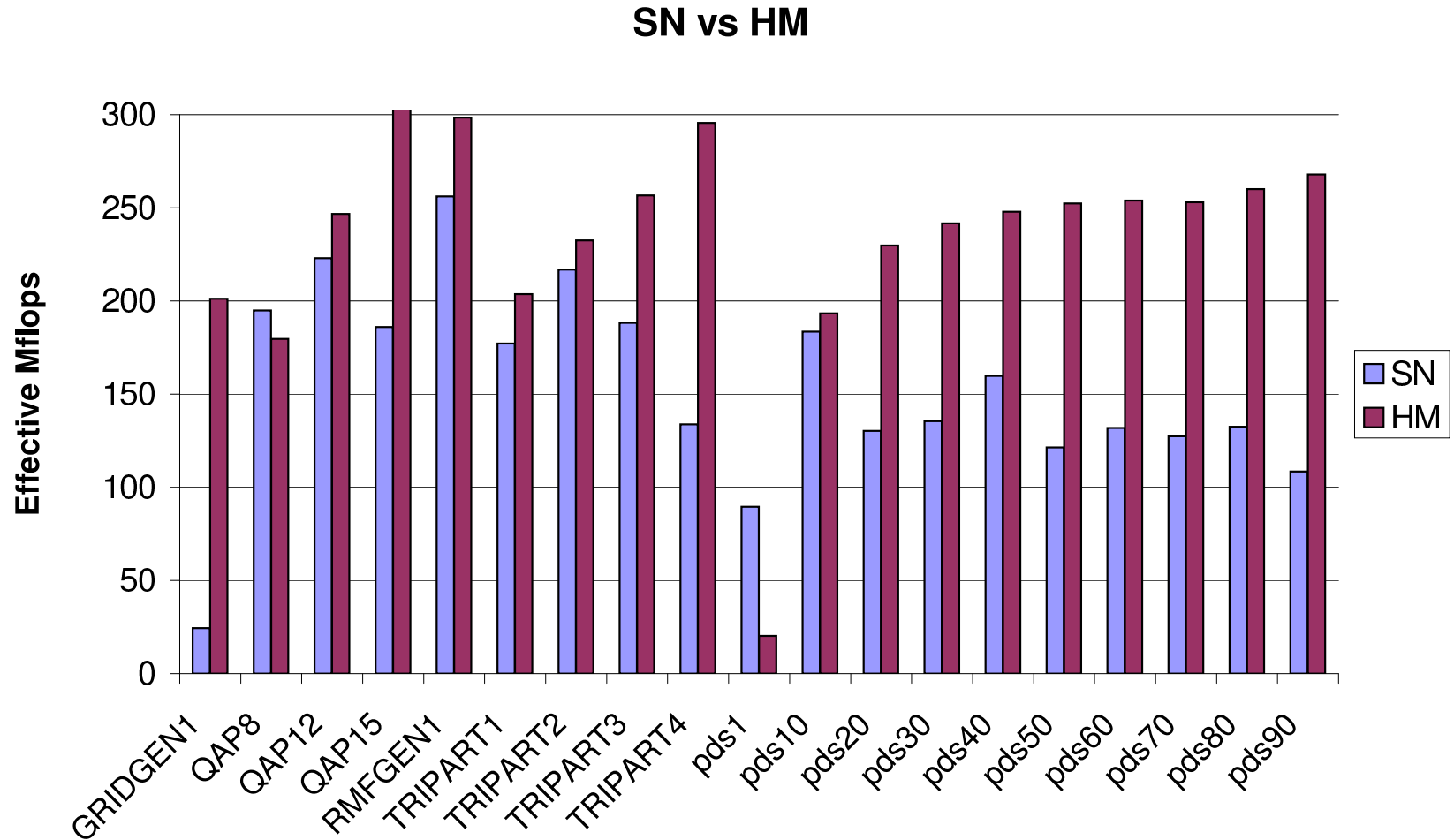


# HM vs SN: PDS matrix family



# HM vs SN performance: summary

---



# Summary

---

Techniques which improve Hypermatrix Cholesky performance:

- Efficient kernels which operate on small data submatrices
  - Rectangular data submatrices
- Windows within data submatrices

Techniques which do not improve its performance:

- Bit Vectors

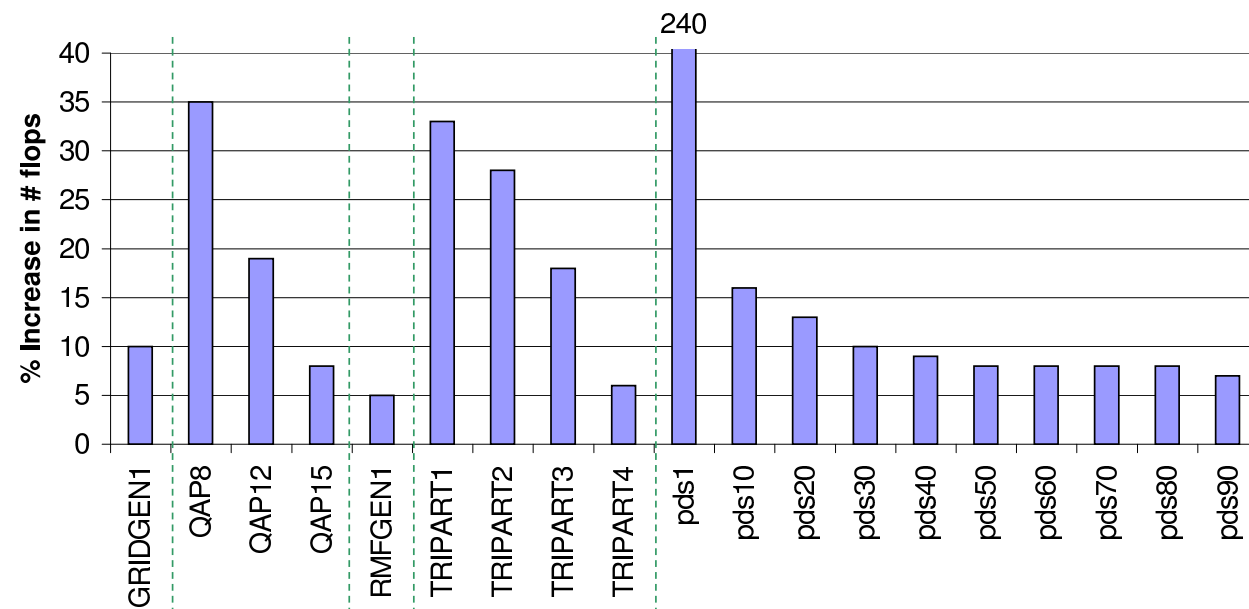
# Conclusions & Future Work

---

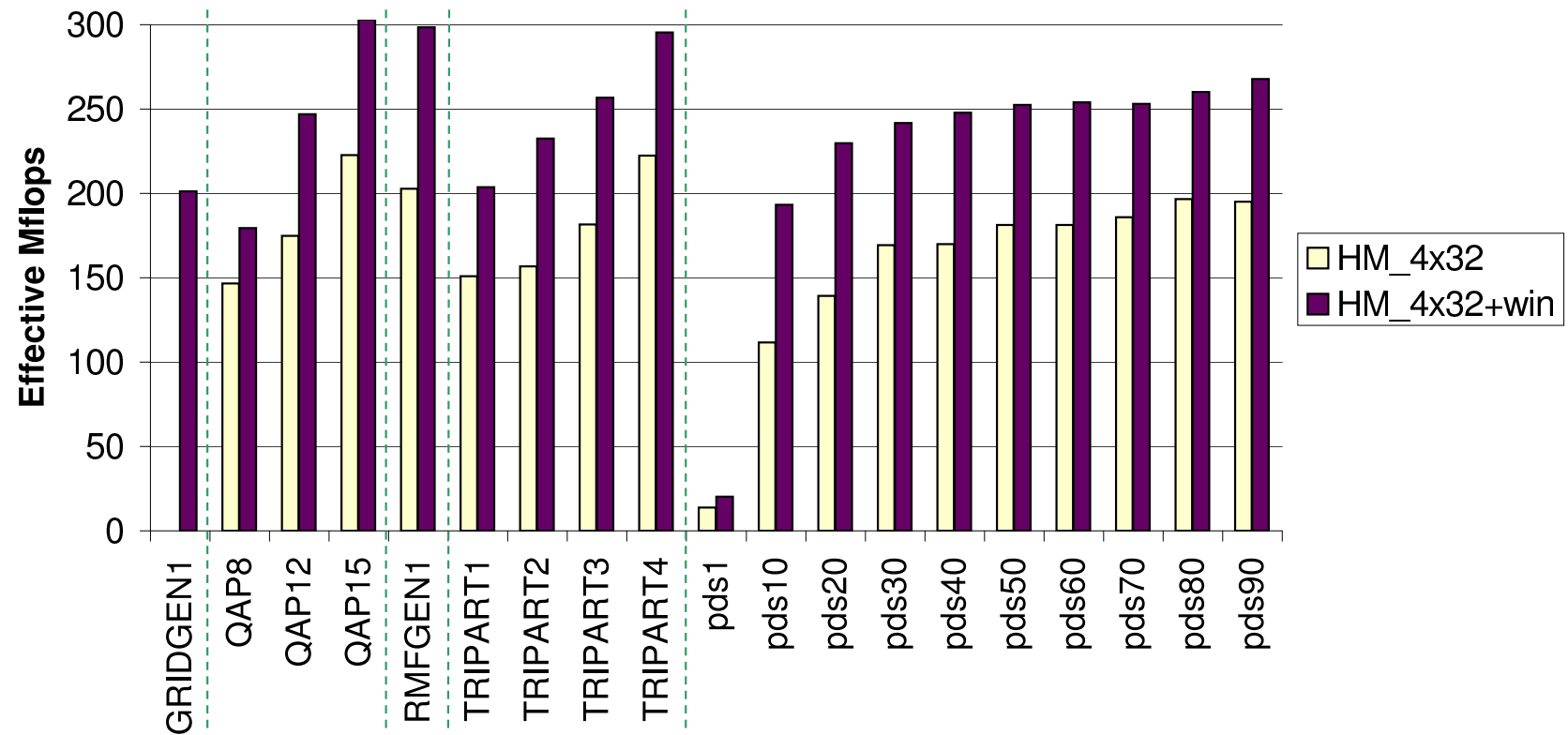
- Sparse HM Cholesky is competitive for large problems
  - OOC
- Good chances for exploiting parallelism
  - Parallel version

# Increase in number of operations in sparse HMI Cholesky (4x32 + windows).

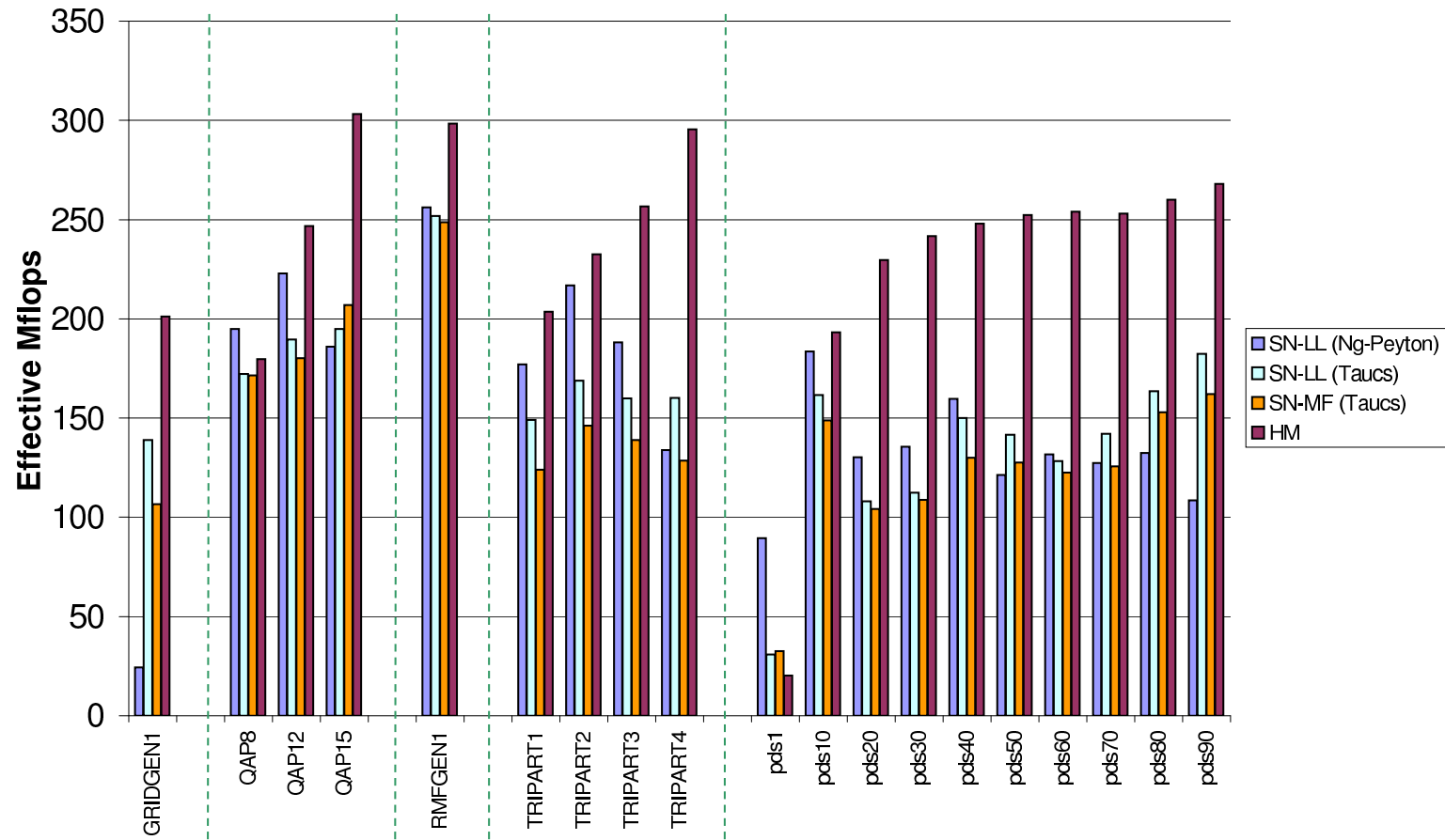
---



# HM performance with and without windows



# Performance of several sparse Cholesky factorization codes.



# Current work

---

- Amalgamation
- Compare with Blocked sparse Cholesky code within SPLASH-2
  - Get it to work for large matrices
  - Get it to work in parallel