

A Study on Load Imbalance in Parallel Hypermatrix Multiplication using OpenMP

José R. Herrero, Juan J. Navarro

{josepr,juanjo}@ac.upc.edu

Computer Architecture Department
Universitat Politècnica de Catalunya



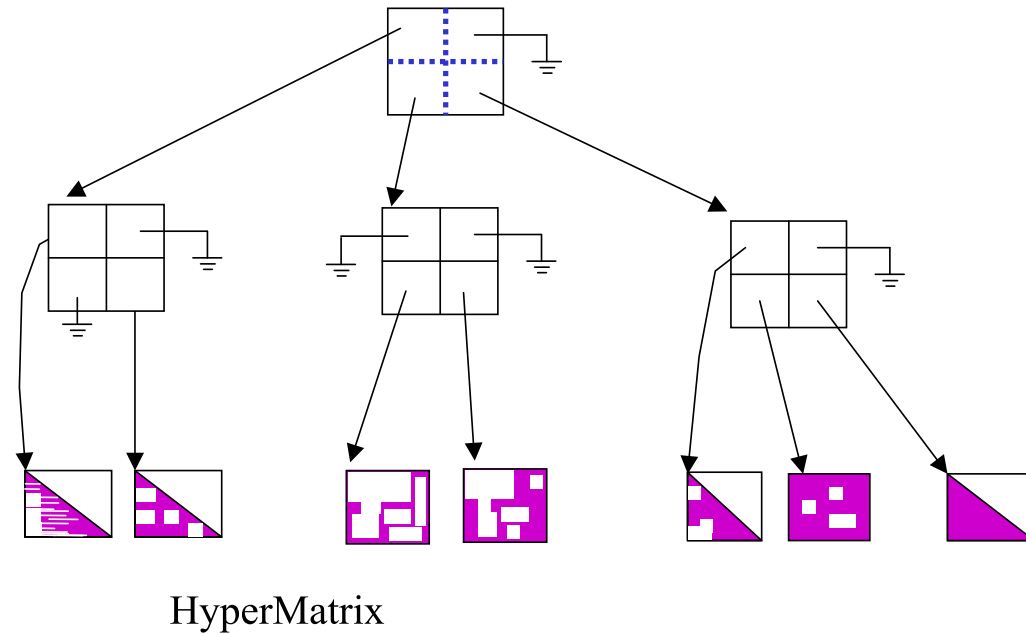
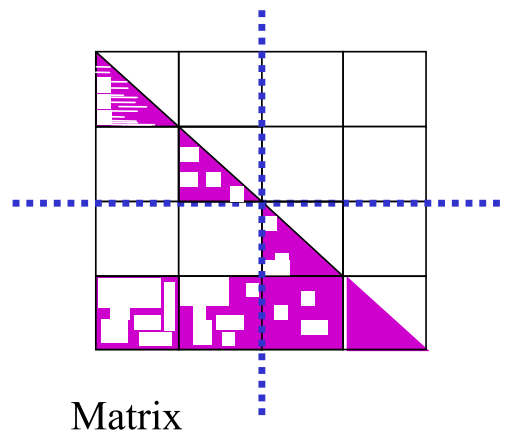
Barcelona

Spain

Outline

- Introduction
 - Hypermatrices
 - Statement of the problem
- Solution attempts
- Conclusions

A sparse matrix and an equivalent Hypermatrix



Getting Efficient Kernels

We fix data submatrix size to get good performance
(Talk given this morning)

Goal: The sparse code requires efficient operation on small data submatrices

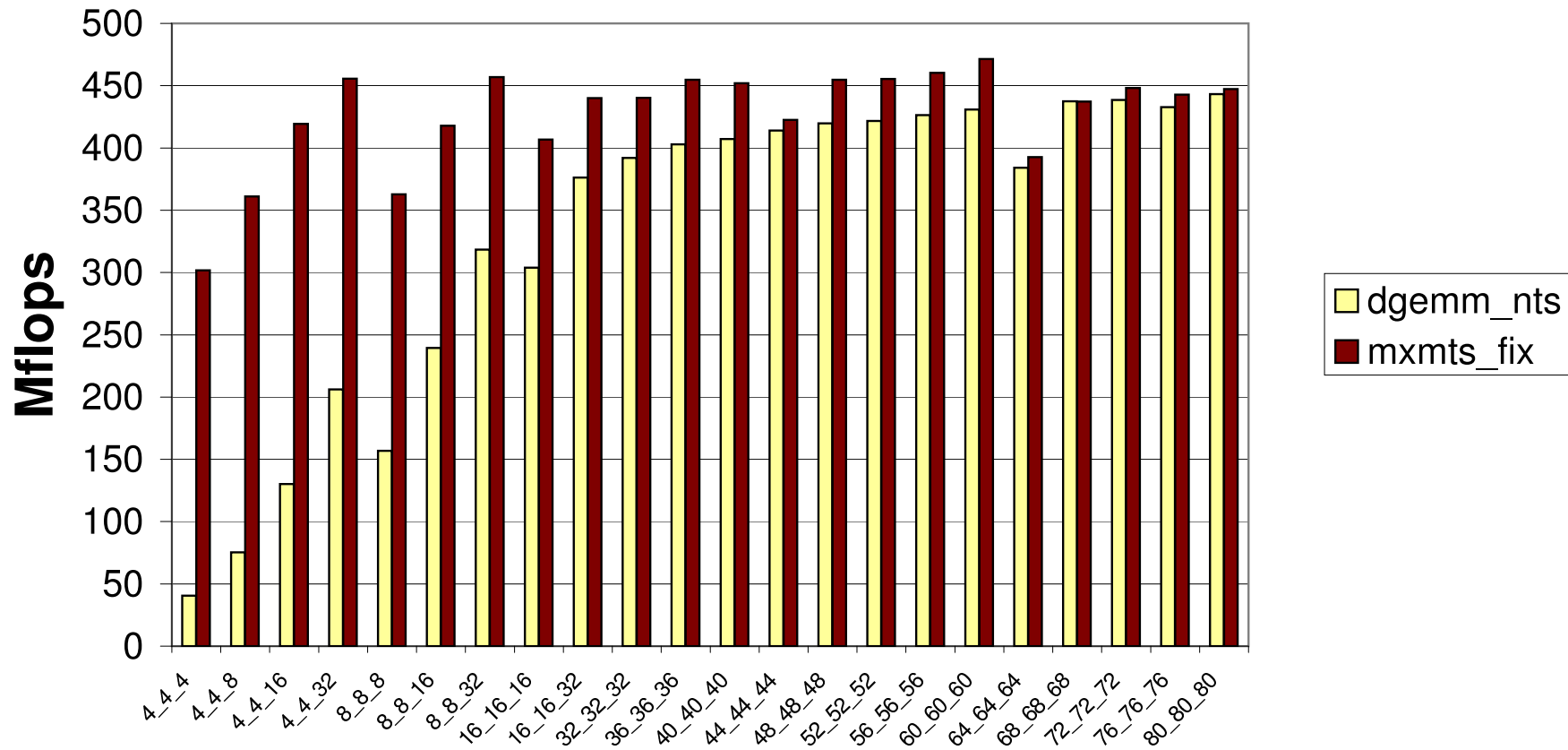
- Reduce data submatrix size while keeping good BLAS3 performance

Idea [Euro-Par'03]:

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

Matrix multiplication performance on small matrices: R10K

$$C = C - A * B^t$$



R10000 250 MHz (500 Mflops peak)

Problem

- We fix data submatrix size to get good performance (also for dense codes)
(Talk given this morning)
 - We force upper levels' dimensions to be multiple of those in lower levels

⇒ Partitioning of upper level pointer matrix driven by dimensions of lower levels

⇒ Does not necessarily suit number of CPUs
- Consequence:
 - **Load imbalance** when **dimension of** the matrix in the **upper pointer level** is low and is **not multiple of the number of processors** used.

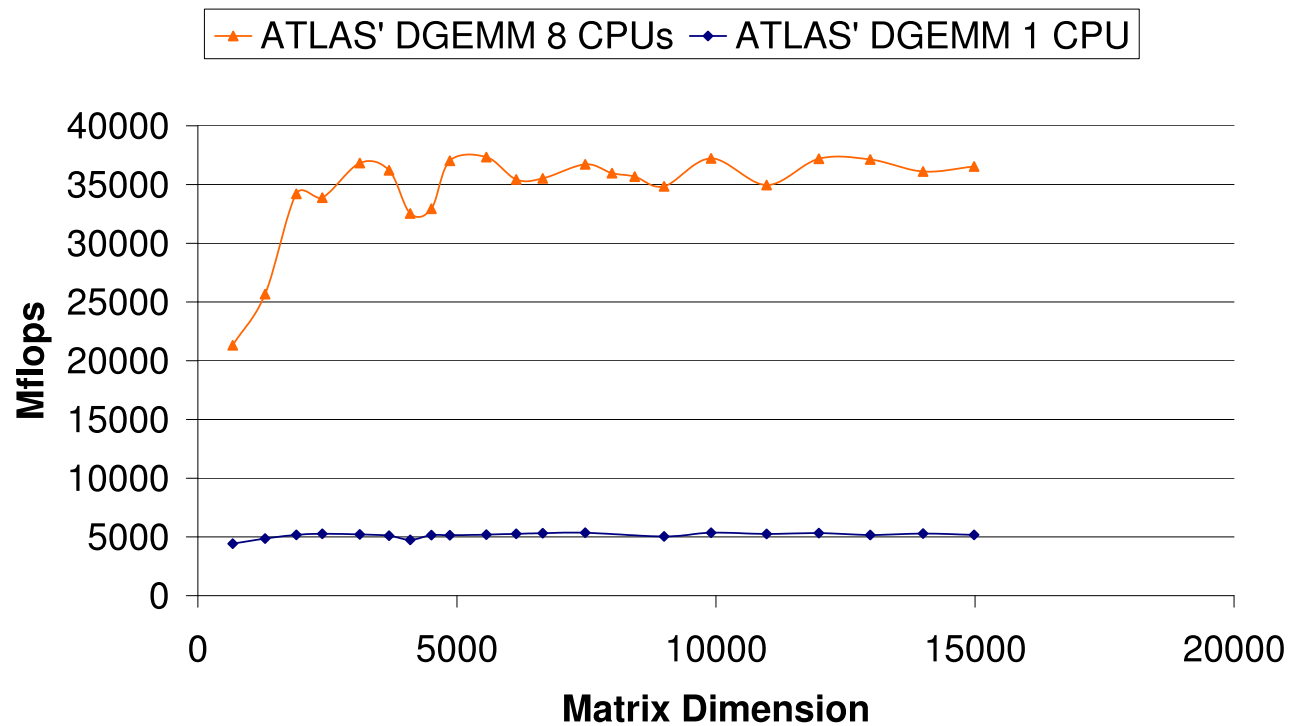
Can the problem be solved/reduced?

- Study a highly parallelizable code
 - Dense Matrix Multiplication
- Try OpenMP features
 - Static/Dynamic Scheduling
 - Chunk size
 - Nested parallelism

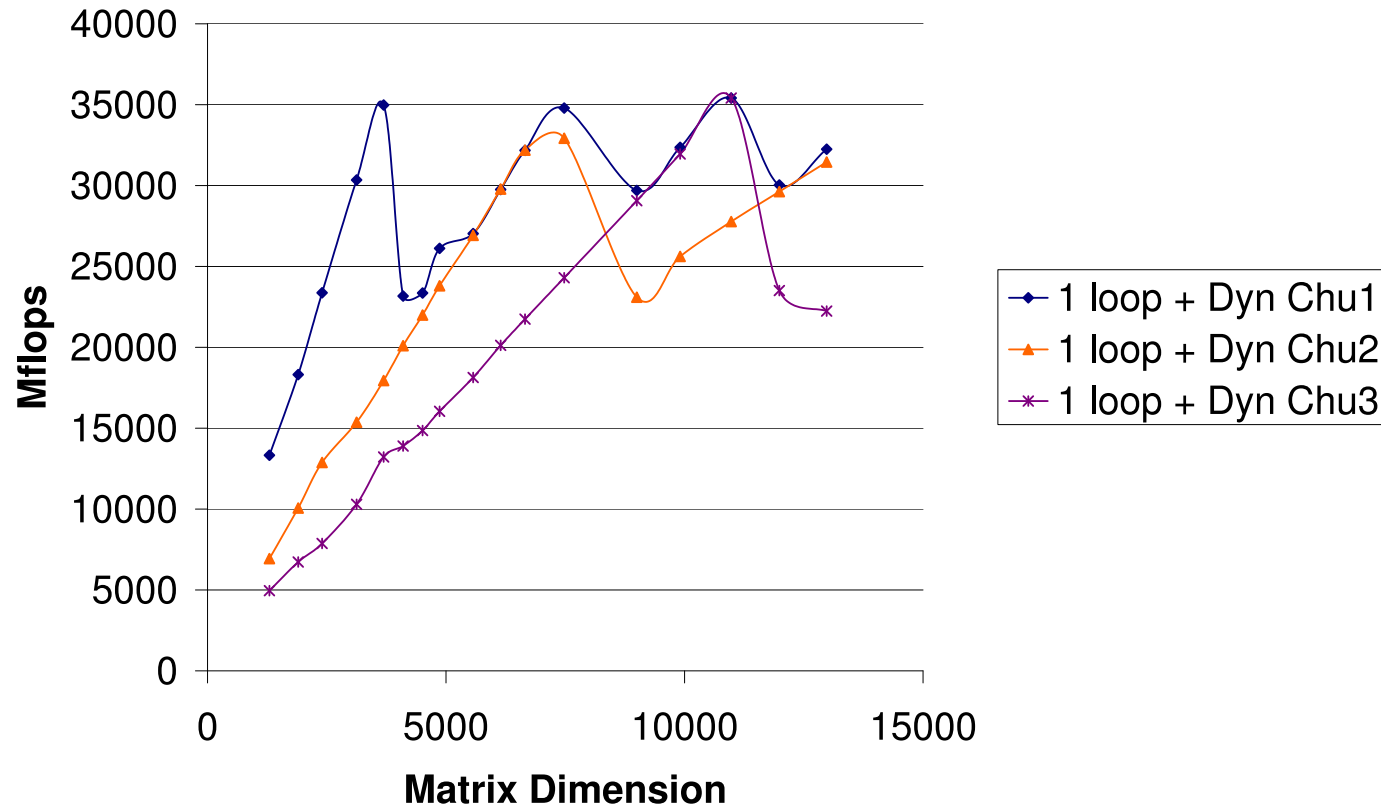
Computing Environment

- Hardware
 - 8-way SMP
 - Itanium2 processors @ 1.5 GHz
 - 48 Gflops (theoretical peak)
- Software
 - Intel Fortran Compiler 8.0
 - * OpenMP

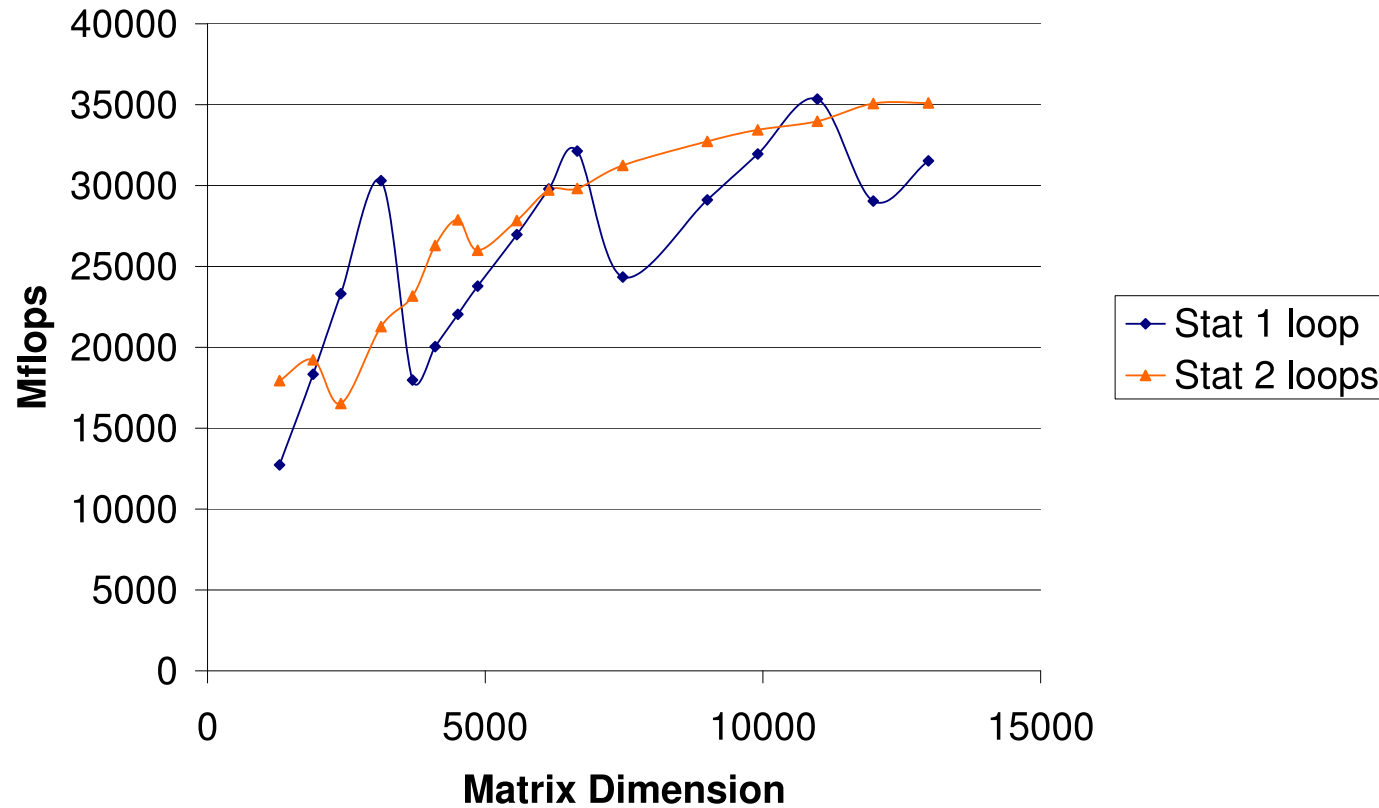
ATLAS Performance



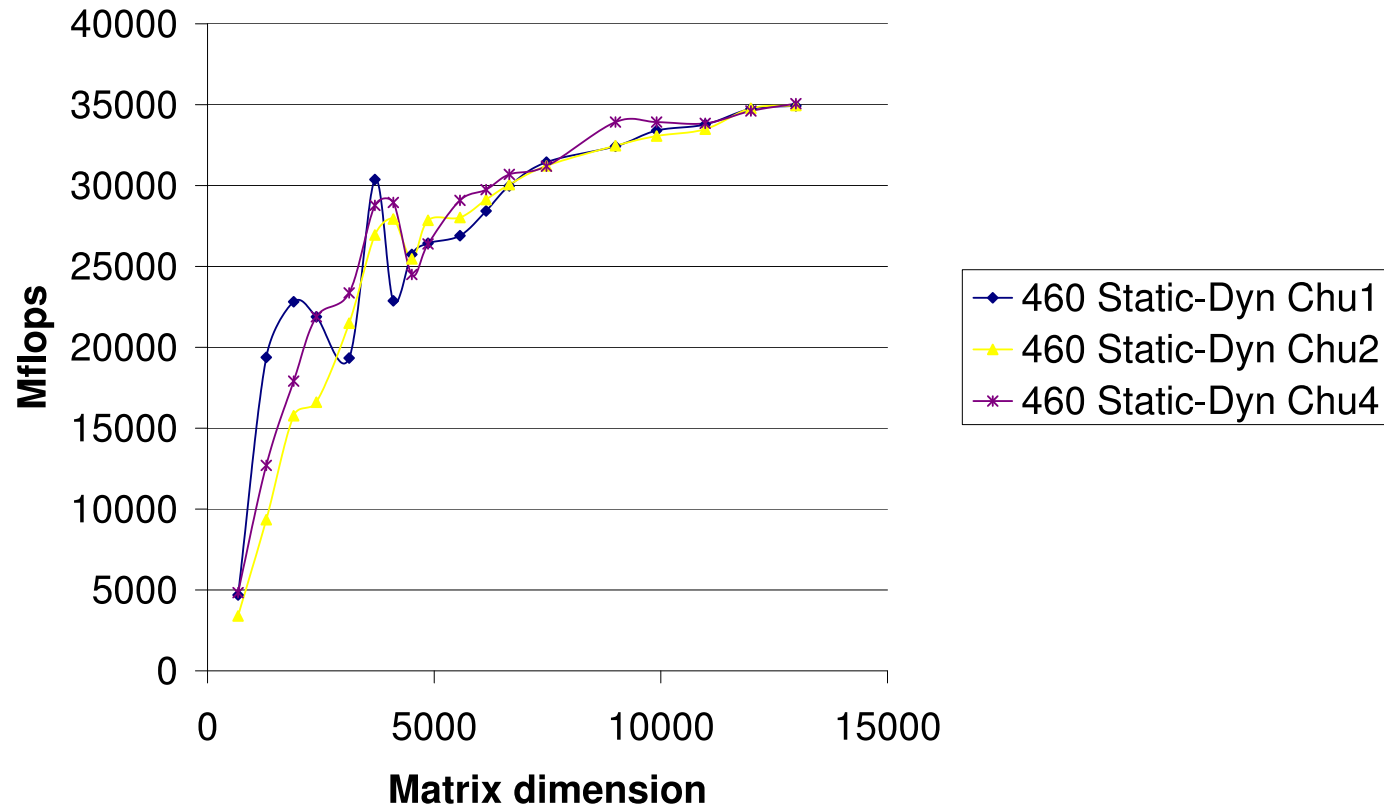
Dynamic Scheduling



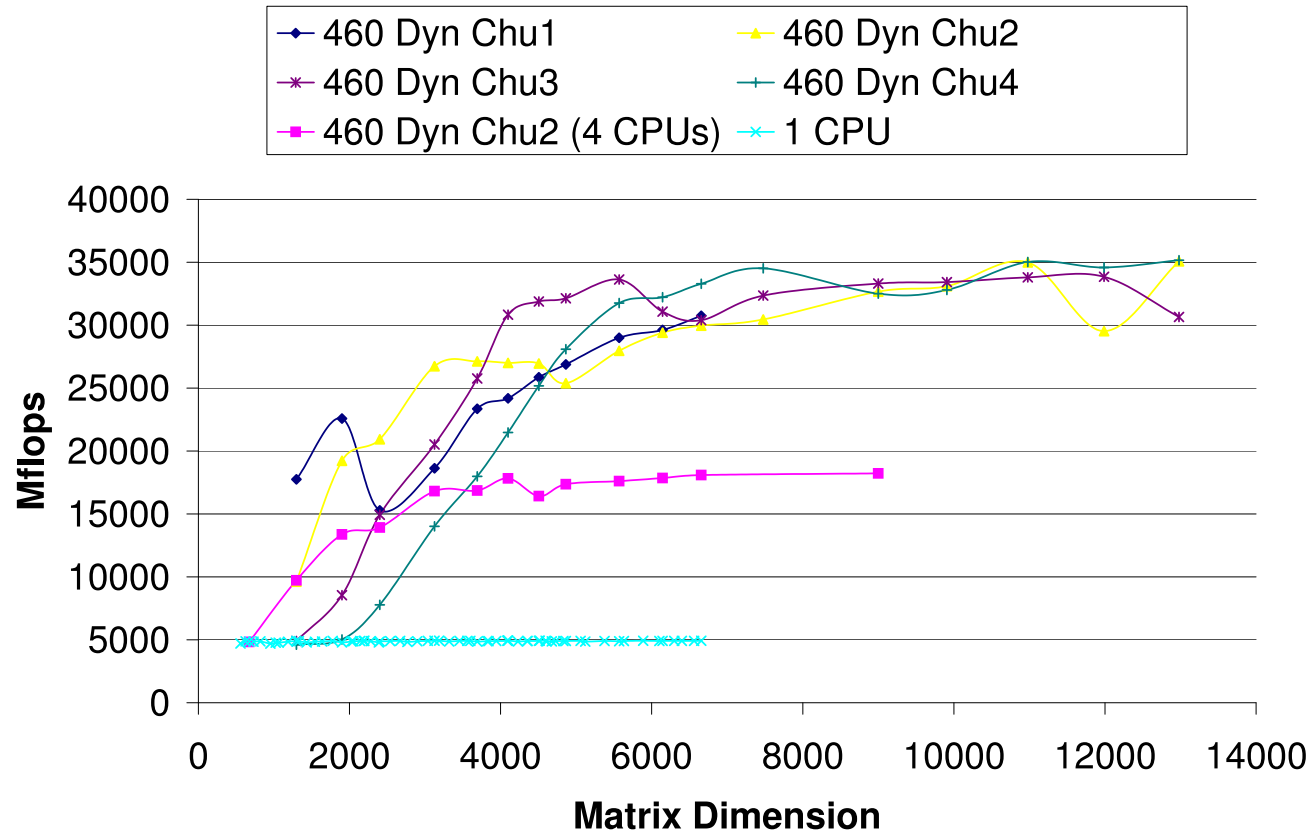
Static Scheduling



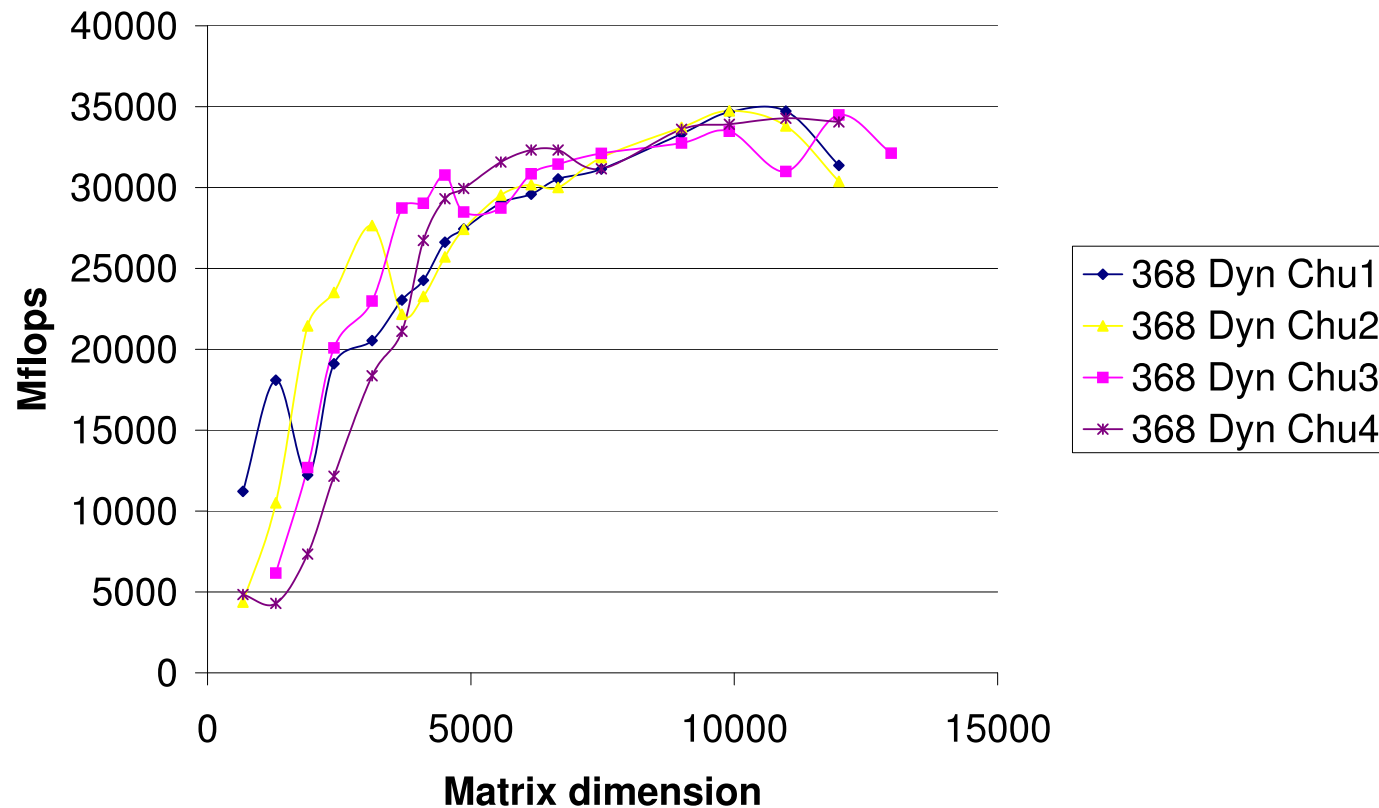
Static + Dynamic Scheduling



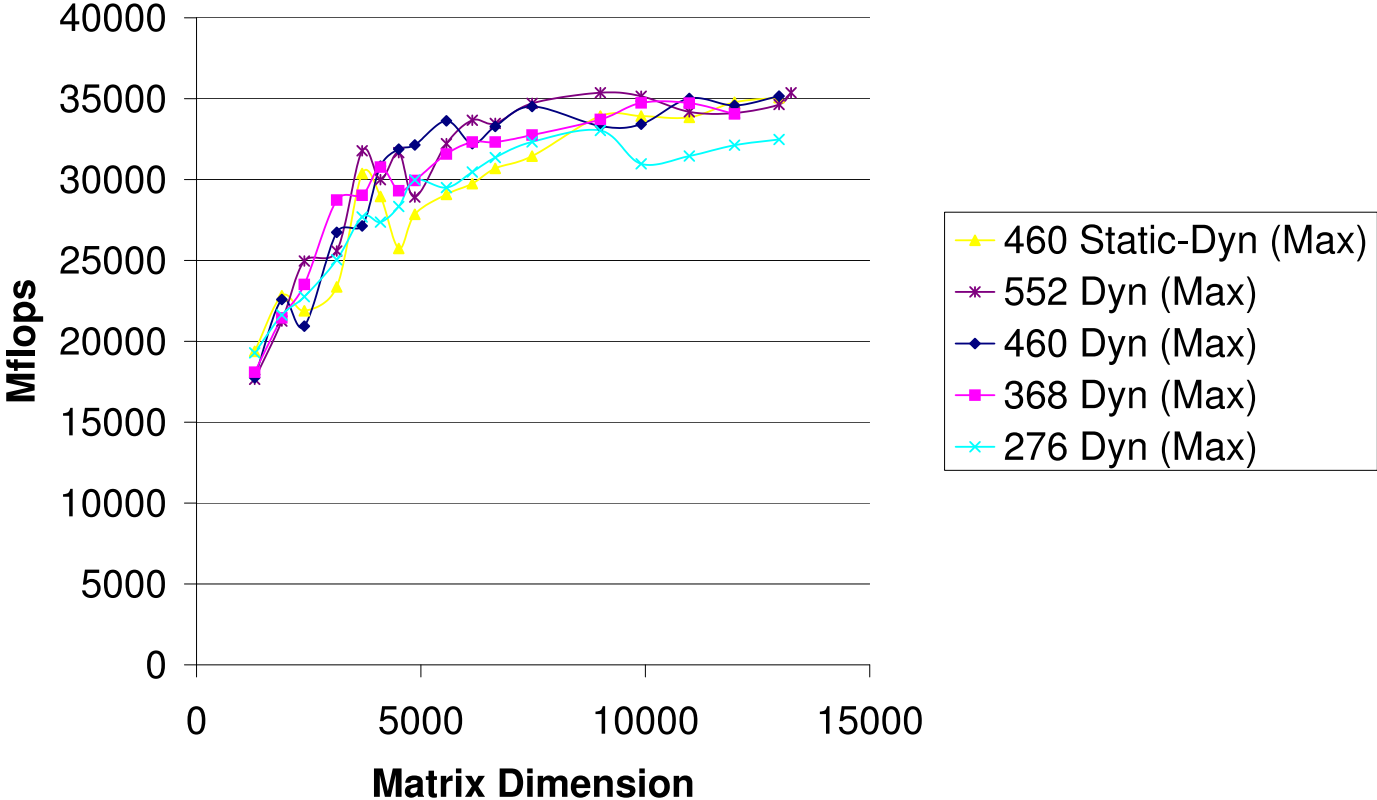
Dynamic Scheduling: 2 nested parallel loops



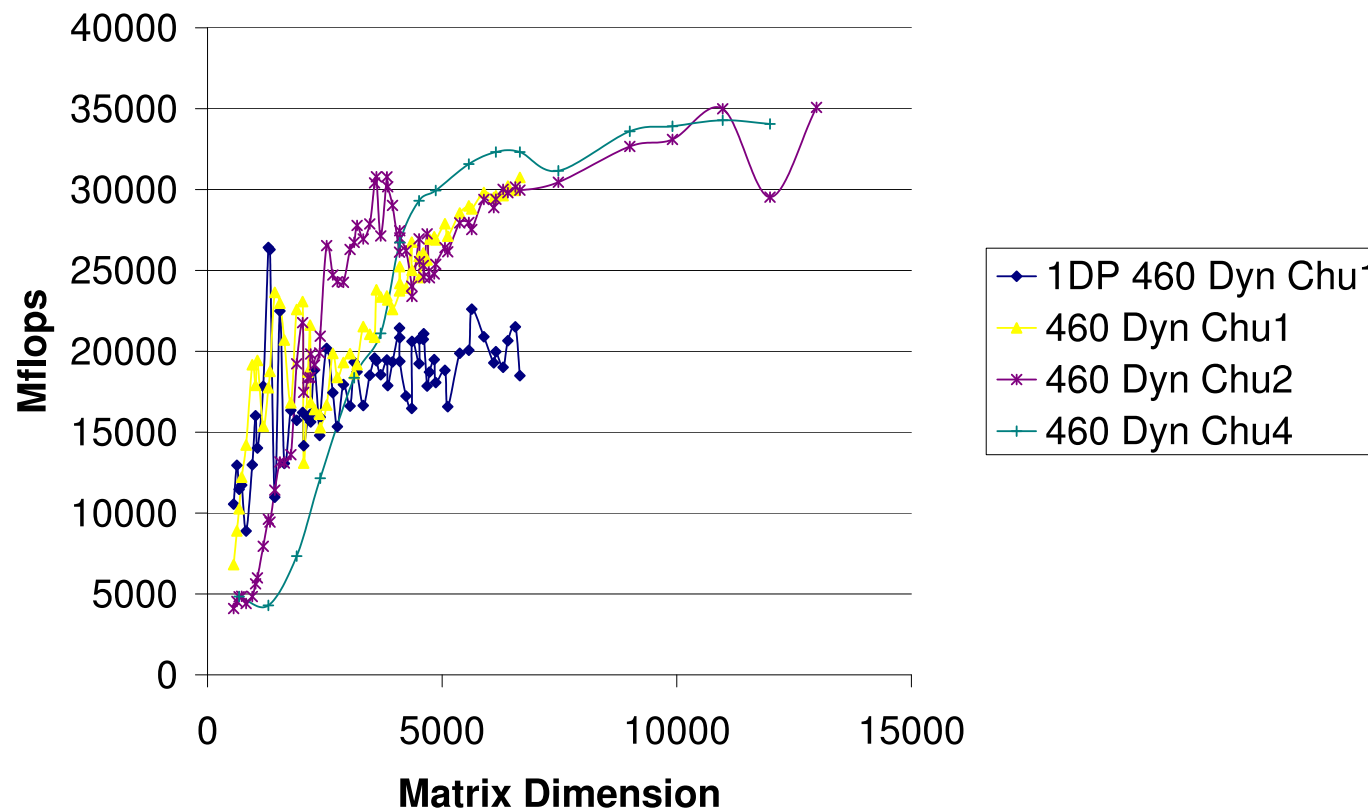
Dynamic Scheduling: smaller block size



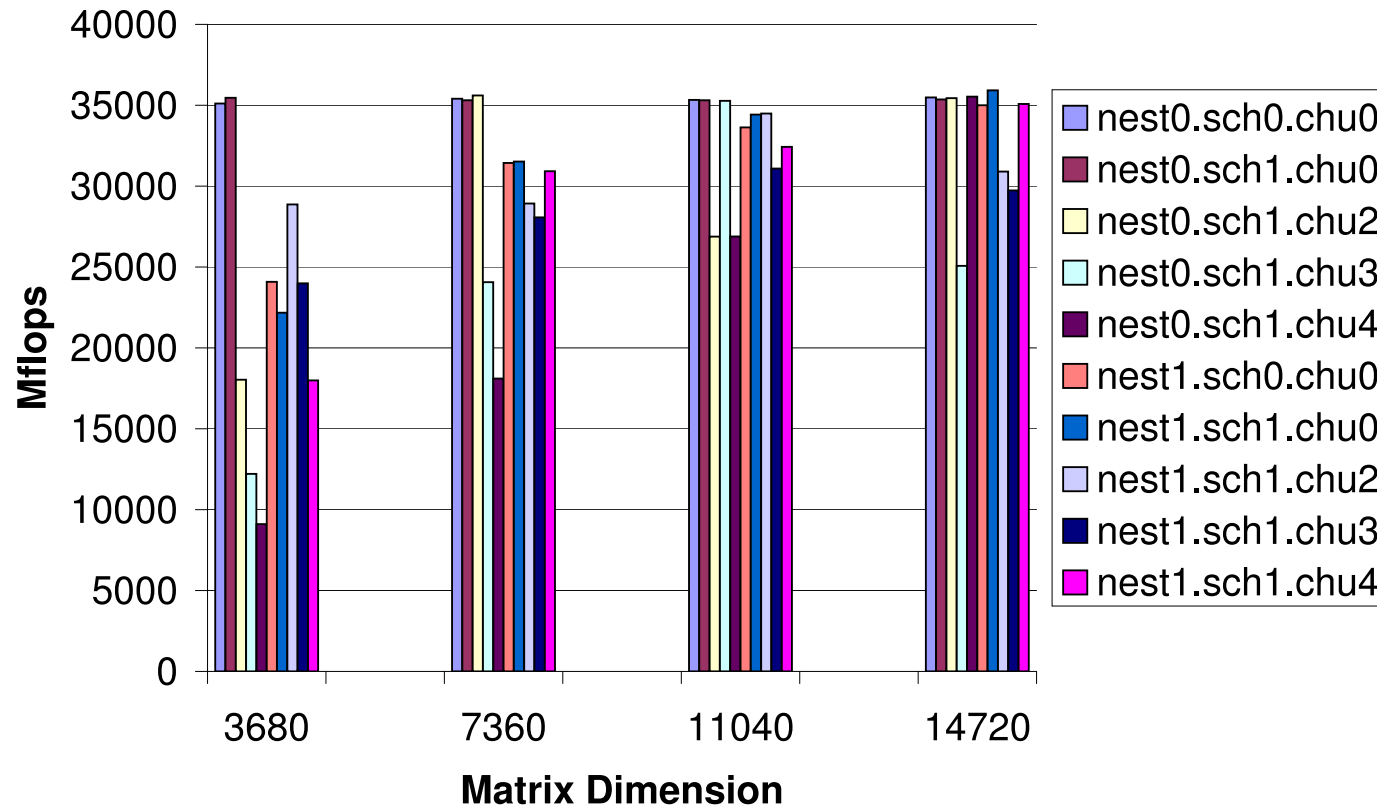
Block Size



3 parallel loops: (includes outermost loop in level of pointers to data)



Perfect Matrix Partitioning



Number of matrix partitions multiple of number of threads

Upper level maps blocks of size 460×460

\Rightarrow # pointers in upper level of HM: 8, 16, 24 and 32 respectively

Conclusions and future work

- Data structures are important! (Once again)
- If data can be partitioned adequately
 - Parallelization of outer loop (static or dynamic) works well
 - Do not need nested parallelism (introduces overhead)
- Else
 - Nested parallelism opens new parallel sections which can use idle processors
- Future work
 - Replace the hypermatrix data structure in dense codes
 - Use a plain storage of the data submatrices which can be accessed with a simple indexing scheme.