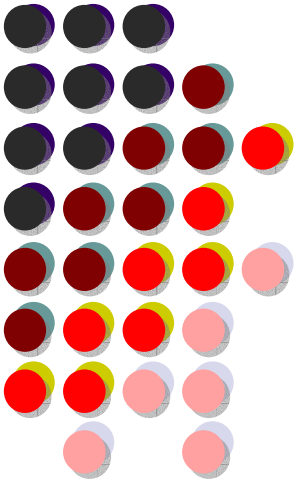
Improving Linear Algebra Computation on NUMA platforms through auto-tuned nested parallelism

Javier Cuenca, Luis P. García, Domingo Giménez

Parallel Computing Group
University of Murcia, SPAIN



Introduction

- Scientific and engineering problems are solved with large parallel systems
- In some cases those systems are NUMA
  - A large number of cores
  - Share a hierarchically organized memory
- Kernel of the computation for those problems: BLAS o similar
  - Efficient use of routines → a faster solution of a large range of scientific problems
- Normally: multithreaded BLAS library optimized for the system is used, but:
  - If the number of cores increases → the degradation in the performance grows
- In this work:
  - Analysis of the behaviour in NUMA of the matrix multiplication of the BLAS
  - Its combination with OpenMP to obtain nested parallelism
  - An auto-tuning method → a reduction in the execution time

Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Computational systems

- **Ben**
  - Part of the system Ben-Arab´ı of the Supercomputing Center of Murcia.
  - A shared-memory system with 128 cores.
  - HP Integrity Superdome with architecture NUMA
  - Hierarchical composition with crossbar interconnection.
  - Each computing node:
    - an SMP with four CPUs dual core Itanium-2
    - an ASIC controller to connect the CPUs with the local memory and the crossbar commuters
  - Access to the memory is non uniform: Four different costs in the access to the shared-memory.

- **Pirineus**
  - A system at the Centre de Supercomputacio de Catalunya.
  - An SGI Altix UV 1000
    - a total of 224 Intel Xeon six-core serie 7500 (1344 cores)
    - An interconnection NUMAlink 5 in a paired node 2D torus.
  - The access to the memory is non-uniform

Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

The software

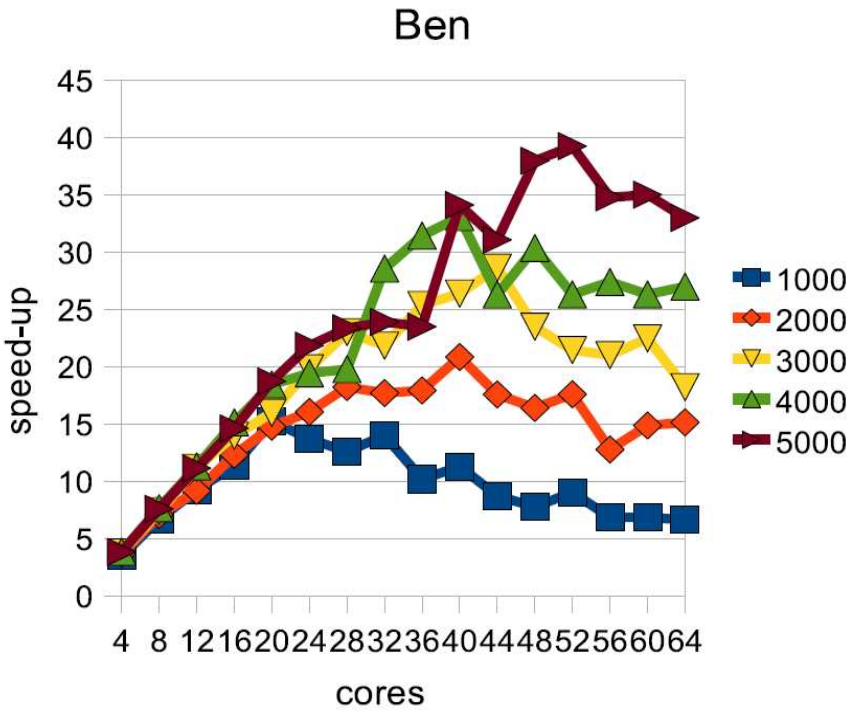
- The matrix multiplication routine used: the double precision routine `dgemm`.
- The BLAS implementation of the Intel MKL toolkit used is the version 10.2
- The libraries are multithreaded: calling the routine with the desired number of threads:
  - If dynamic parallelism is enabled → the number of threads is decided by the system (less than or equal to that established).
- The C compiler used was Intel icc version 11.1 in both platforms.
- Two-level parallelism:
  - a number of OpenMP threads + calls to the multithreaded BLAS
- Matrices A and B can be multiplied with two-level parallelism:
  - *q* threads OpenMP
  - each thread multiplying a block of adjacent rows of matrix A by the matrix B
  - establishing a number of threads (*p*) to be used in the matrix multiplication in each OpenMP thread

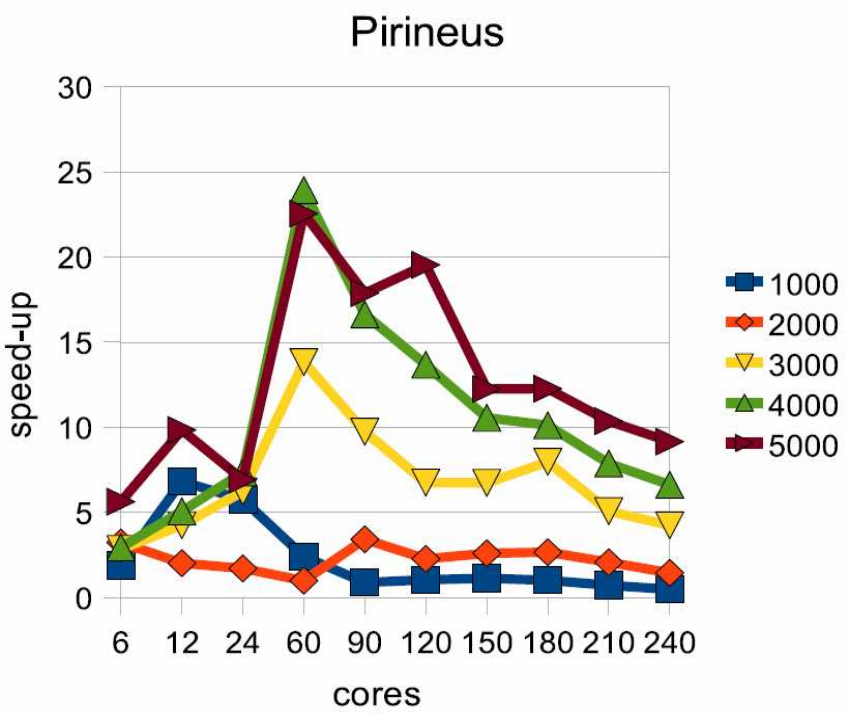
Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Motivation

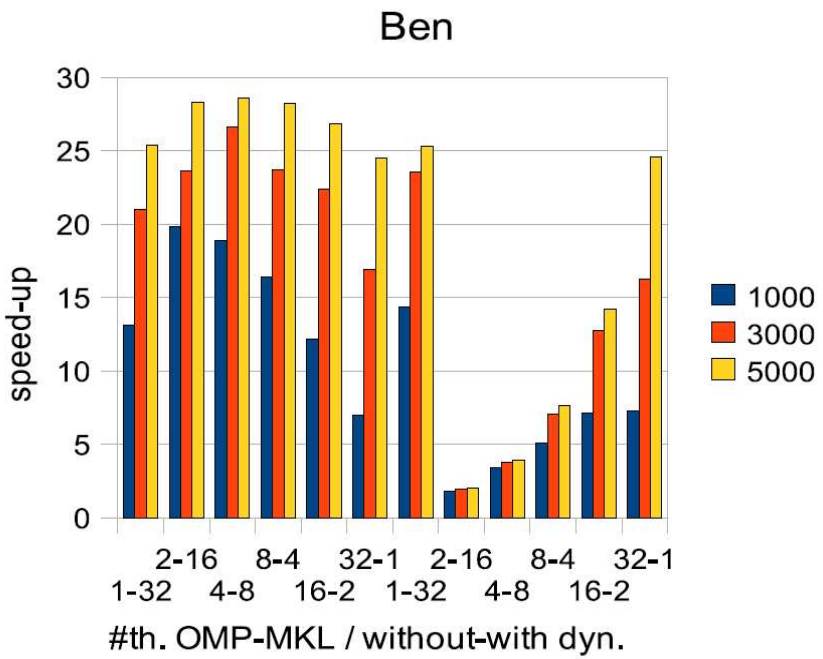
- Using a multithreaded version of BLAS → the `dgemm` MKL routine
- The optimum numbers of threads changes from one platform to another and for different problem sizes.
- Default option (number of threads = available cores) is not good

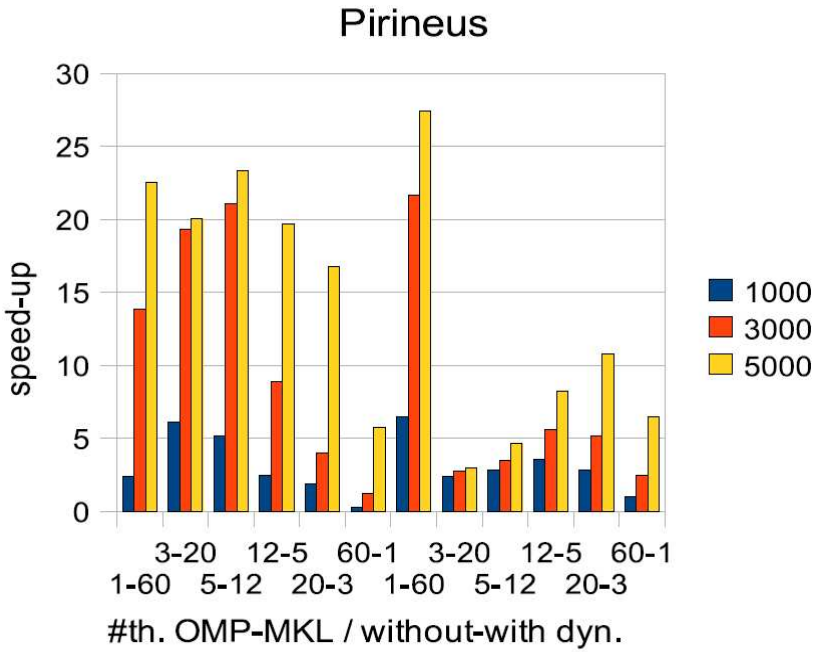




Motivation

- **Dynamic Selection of threads:**
  - Reduction in the speed-up increases with the number of OpenMP threads
  - Number of MKL threads used is just one
- **No Dynamic Selection of threads:**
  - bigger speed-ups are obtained
  - Number of OpenMP threads grows → an increase of the speed-up until a maximum
  - So, a large number of cores → a good option to use a high number of OpenMP threads





Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Automatic optimisation method

- Automatic Tuning System (ATS) focused on modelling the execution time

$$T_{exe} = f(n,SP,AP)$$

  - ***n***: the problem size
  - ***SP***: System Parameters. Characteristics of the platform (hardware + basic installed libraries)
  - ***AP***: Algorithmic Parameters. Values chosen by the ATS to reduce the execution time

- An adaptation to large NUMA platforms:
  - Each arithmetic operation: data access time depends on the relative position in memory space
    - Data can be in the closest memory of the processor or in that of another processor
    - The interconnection network could be non homogeneous
  - Therefore
    - → those data could be at different distances from the processor that needs them
    - → the access time is modelled with a hierarchical vision of the memory
  - It is also necessary to take into account the migration system of the platform

Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Automatic optimisation method

Design phase: modelling the execution time of the routine

Modelling 1-Level: MKL multithreading dgemm without generating OpenMP threads

- Model:

$$T_{dgemm} = \frac{2n^3}{p} k_{dgemm}$$

- *AP*: p → Number of threads inside the MKL routine dgemm
- *SP*: k_{dgemm} → time to carry out a basic operation inside the MKL routine dgemm (including memory acceses). Taking into account the data migration system:

$$k_{dgemm} = \alpha k_{dgemm\_NUMA}(p) + (1-\alpha) k_{dgemm\_M_1}$$

- $k_{dgemm\_M\_1}$: operation time when data are in the clostest memory to the operating core
- k_{dgemm_NUMA}: operation time when data are in any level of the RAM memory

- α: weighting factor
  - directly proportional to the use by each thread of data assigned to the other (*p-1)* threads
  - inversely proportional to the reuse degree of data carried out by the routine (dgemm)

$$\alpha = min\left\{1, \frac{p(p-1)}{n^3 / n^2}\right\}$$

Automatic optimisation method

Design phase: modelling the execution time of the routine

Modelling 1-Level: MKL multithreading dgemm without generating OpenMP threads

- Platform:
  - H memory levels
  - c_l cores have a similar access speed to the level l, with $1 \leq l \leq H$

- k_{dgemm_NUMA} value can be modelled, depending on p:
  - If $0 < p \leq c_1$:

$$k_{dgemm\_NUMA}(p) = k_{dgemm\_M_1}$$

  - else if $c_1 < p \leq c_2$:

$$k_{dgemm\_NUMA}(p) = \frac{c_1 k_{dgemm\_M_1} + (p - c_1) k_{dgemm\_M_2}}{p}$$

  - …, in general, if $c_{H-1} < p \leq c_H$:

$$k_{dgemm\_NUMA}(p) = \frac{\sum_{l=0}^{H-2} (c_l - c_{l-1}) k_{dgemm\_M_l} + (p - c_{L-1}) k_{dgemm\_M_L}}{p}$$

Automatic optimisation method

Design phase: modelling the execution time of the routine

Modelling 2-Level: OpenMP threads + MKL multithreading dgemm

- Model:

$$T_{2L\_dgemm} = \frac{2\frac{n}{q}nn}{p} k_{2L\_dgemm} = \frac{2n^3}{R} k_{2L\_dgemm}$$

- *AP*: ***R=pxq* threads** interactuating
  - *p* → Number of threads inside the MKL routine dgemm
  - *q* → Number of OpenMP threads

- *SP*: k_{2L_dgemm} → time to carry out a basic operation

$$k_{2L\_dgemm} = \alpha k_{2L\_dgemm\_NUMA}(R,p) + (1-\alpha) k_{2L\_dgemm\_M_1}$$

$$k_{2L\_dgemm\_NUMA}(R,p) = \frac{k_{dgemm\_NUMA}(R) + k_{dgemm\_NUMA}(p)}{2}$$

$$\alpha = min\left\{1, \frac{R(R-1)}{n^3/n^2}\right\}$$

Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Automatic optimisation method

Installation phase: experimental estimation of the *SP* values

- General process: calculating the *SP* values that appear in the model
- *SP* values to calculate: k_{dgemm_M1} ,…, $k_{dgemm\ ML}$
- For each memory level l, $1 \leq l \leq H$:

1. Executing `dgemm` → experimental execution time:
   - for a fixed (preferably small) problem size, n
   - for a number of threads, p_l, with $c_{l-1} < p_l \leq c_l$

2. This experimental execution time
   routine model
   α value

   → k_{dgemm_NUMA} for p_l

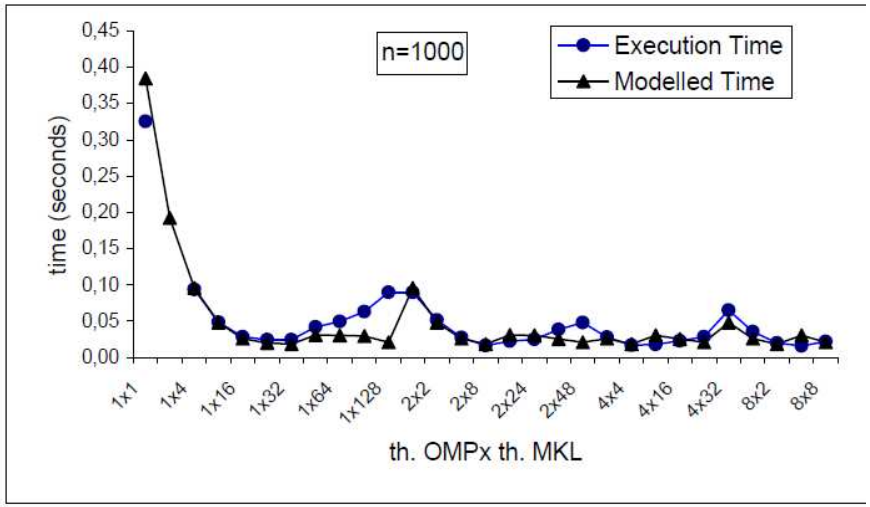
3. k_{dgemm_NUMA} for p_l
   k_{dgemm_NUMA} model
   values of k_{dgemm_M1} ,…,k_{dgemm_Ml-1}

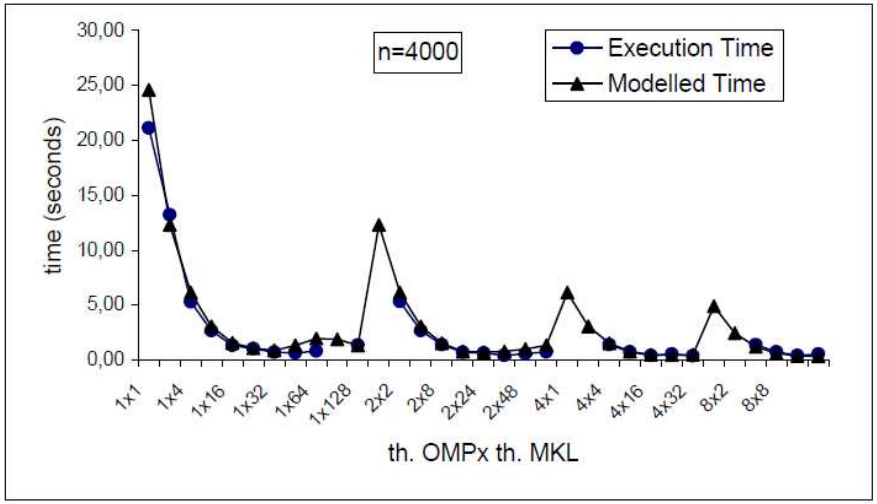
   → k_{dgemm_Ml}

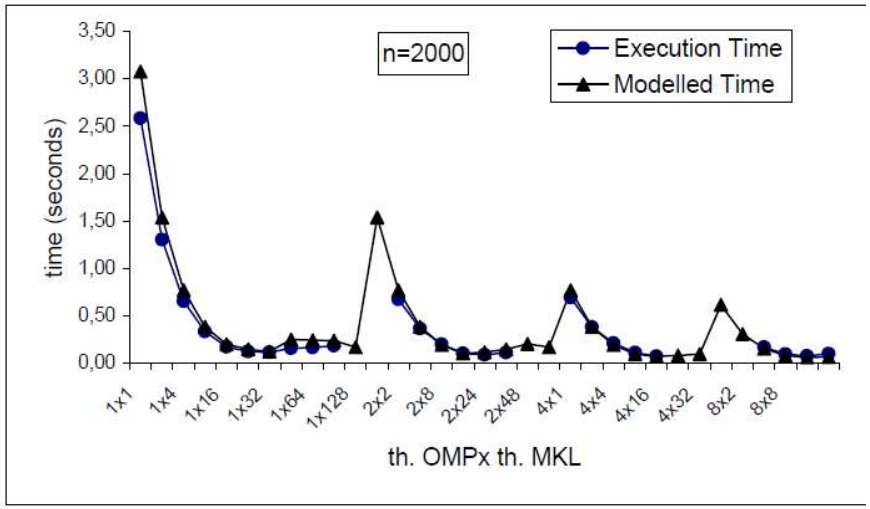
Automatic optimisation method

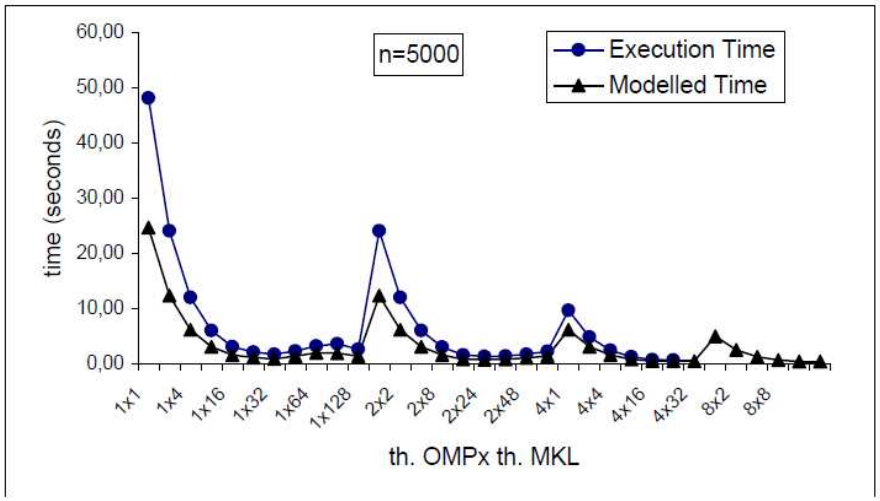
Installation phase: experimental estimation of the *SP* values

Comparison execution vs. modelled time in platform Ben





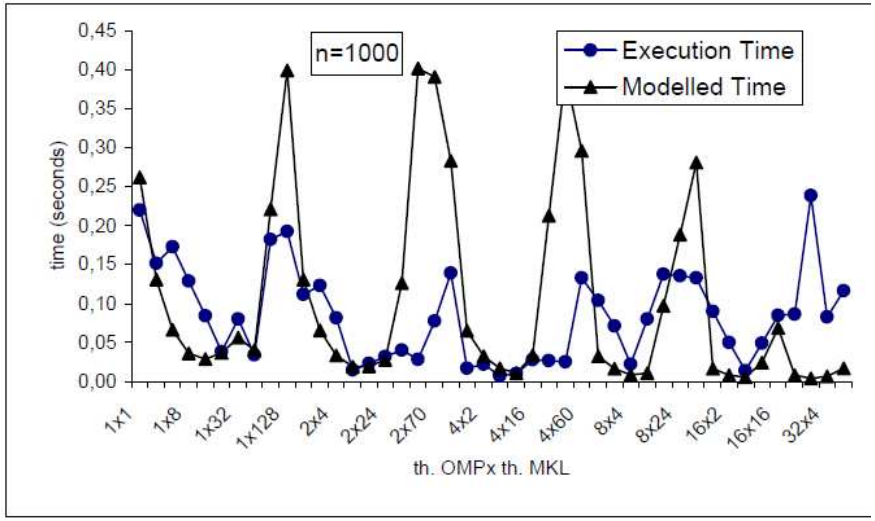


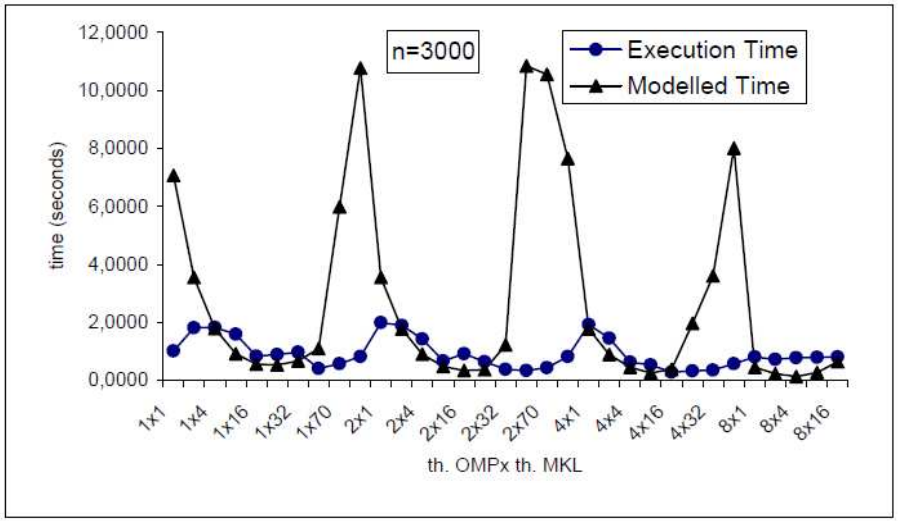


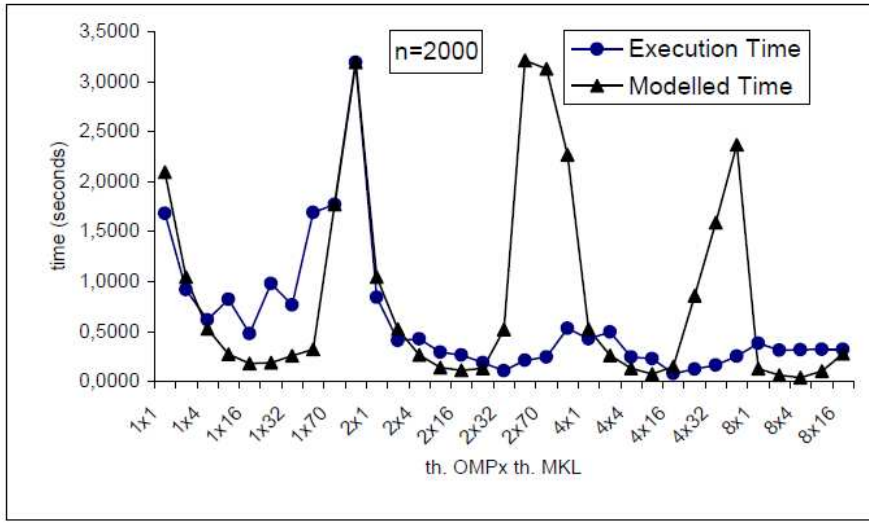
Automatic optimisation method

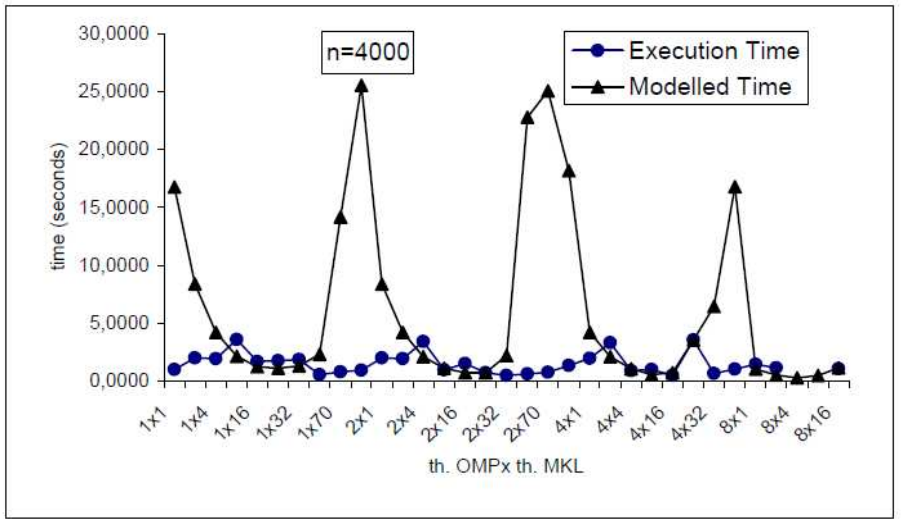
Installation phase: experimental estimation of the *SP* values

Comparison execution vs. modelled time in platform Pirineus









Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Automatic optimisation method

Execution phase: Selection of the *AP* values

- To solve a problem with size *n* in a concrete platform:
  - The ATS takes the model of the routine, the *SP* values calculated for this platform and the value *n*, and selects directly the most appropriate values for the *AP* (number of OpenMP threads, *q*, and MKL threads, *p*)

size	SEQ	MIN-MKL	MC-MKL	AUTO
Ben				
1000	0.320	0.024	0.091	0.012 (2×8)
2000	2.60	0.12	0.39	0.07 (4×16)
3000	8.60	0.32	0.82	0.23 (4×16)
4000	20.22	0.59	1.40	0.74 (4×32)
5000	40.23	1.12	2.11	1.44 (4×32)
Pirineus				
1000	0.224	0.034	0.441	0.021 (16×4)
2000	1.74	0.48	1.19	0.25 (8×8)
3000	5.46	0.39	1.31	0.39 (8×8)
4000	13.14	0.54	1.89	0.95 (8×8)
5000	25.12	1.13	2.65	1.02 (8×16)

Outline

- Introduction
- Computational systems
- The software
- Motivation
- Automatic optimisation method
  - Design phase
  - Installation phase
  - Execution phase
- Conclusions and future work lines

Conclusions and future work lines

- Behaviour of MKL matrix multiplication analysed in 2 NUMA platforms
- Number of threads equal to number of cores: Not always the best option
- Big problems in Large Systems → OpenMP+MKL is a good option
- So, a reduction in the execution time of scientific codes
  - intensively use matrix multiplications or linear algebra routines based on them
  - adequately selecting the threads to be used in the solution of the problem
- This selection: performed automatically by the auto-tuning system
  - Using a model of the execution time of each routine for each platform.
- Future:
  - Same methodology applied to other routines in linear algebra libraries
  - Different numbers of threads in different parts of the program
  - Multi-fabric libraries: routines run differently, depending on the problem