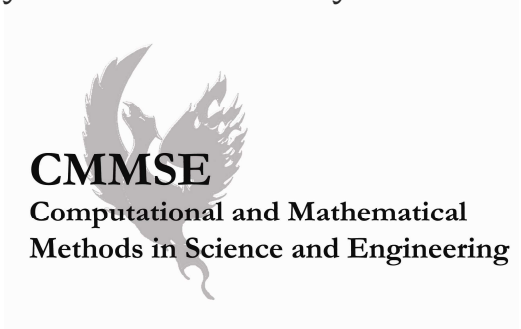
Application of Auto-Tuning Techniques to High-Level Linear Algebra Shared-Memory Subroutines

Jesús Cámara, Javier Cuenca, Domingo Giménez
University of Murcia

Antonio M. Vidal
Polytechnic University of Valencia



July 2012, La Manga, Murcia, Spain

Content

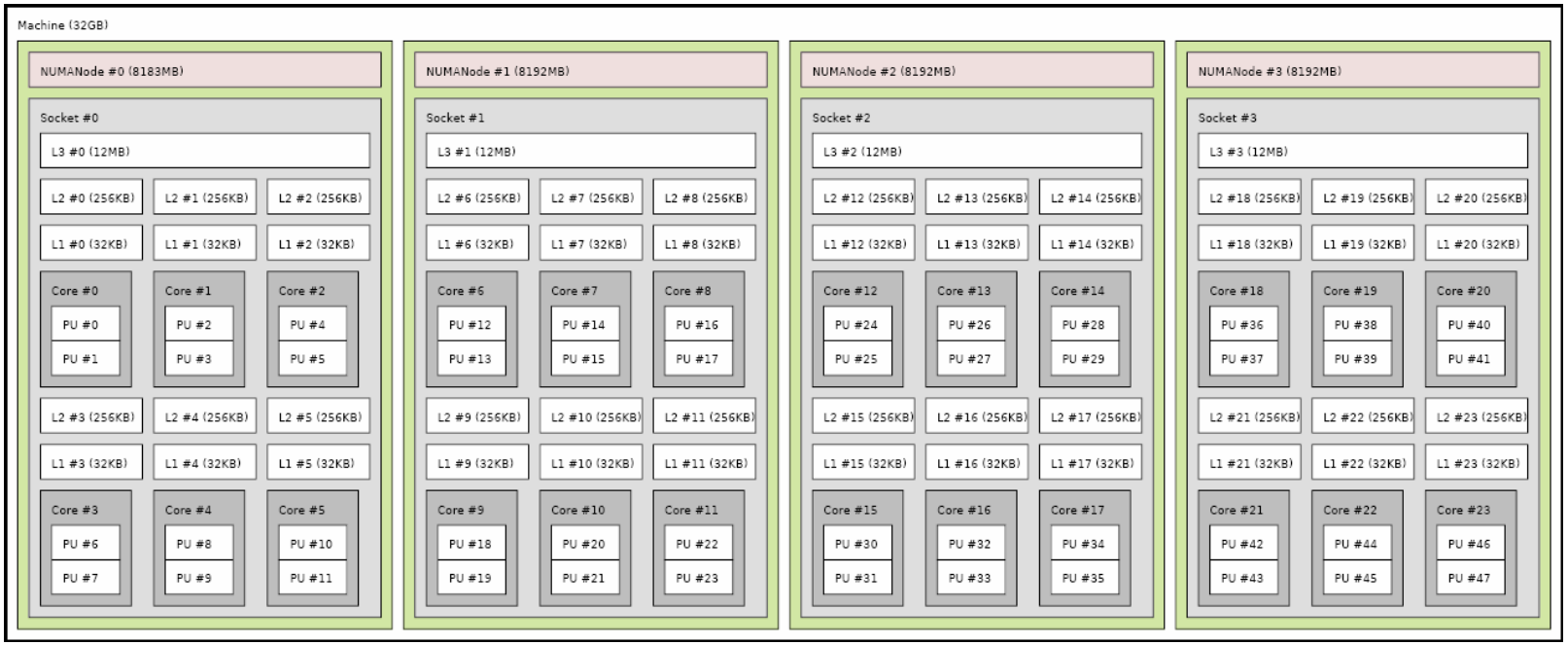
- Introduction
- Multicore Systems
- Two-Level Routines
- The Auto-Tuning Methodology
- Application to Linear Algebra Subroutines
  - Matrix Multiplication Routine (`gemm`)
  - Cholesky Decomposition Routine (`potrf`)
- Conclusions
- Future Work

Introduction

- Large scientific problems are solved in large computational systems using linear algebra libraries (BLAS, LAPACK...)
- Multithread implementations (PLASMA, MKL) are designed:
  - To obtain high performance in multicore and cc-NUMA systems.
  - To select the value of some execution parameters (number of threads, block size...) to reduce the execution time.
- cc-NUMA: the use of the memory hierarchy levels is not an easy task and the performance of multithread implementations decreases when the number of cores increases.

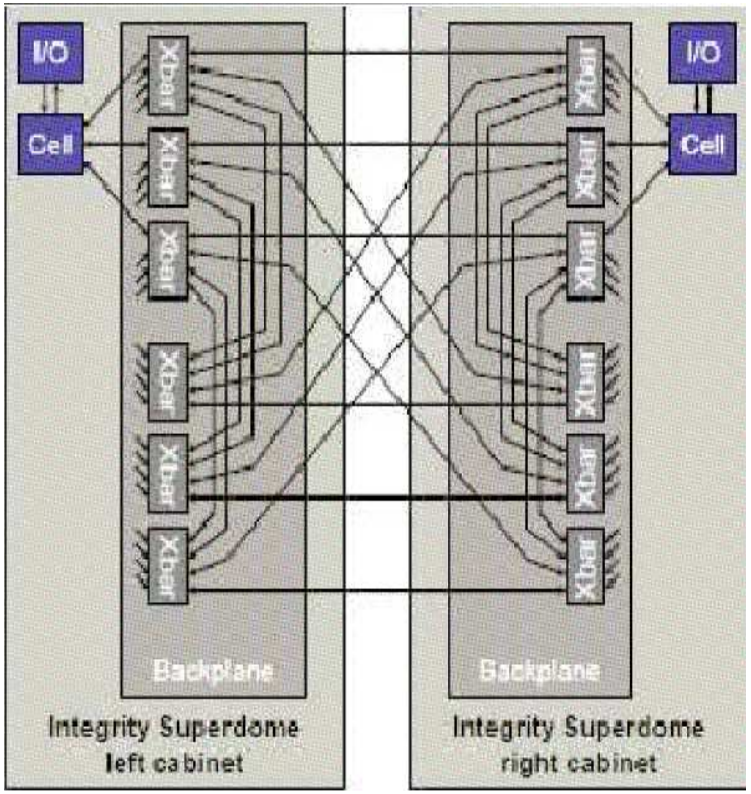
Multicore Systems

- **Saturno** (Computing Science School, University of Murcia)
  - NUMA system with 24 cores (4 nodes with 6 cores each), Intel Xeon E7350 processors, 1.87 GHz. 32 GB of shared-memory and 3 levels of cache (L1, L2 and L3) in each node: the first two levels (L1, L2) for each core and L3 shared by all.



Multicore Systems

- **Ben** (Supercomputing Centre of Murcia):
  - cc-NUMA with 128 cores, Intel Itanium 2 Dual-Core processors, 1.6 GHz, 1.5 TB of shared-memory.
  - Hierarchical composition with crossbar interconnection.
  - Components: the computers and two backplane crossbars.
  - Each computer: 4 dual-core Itanium and a controller to connect the CPUs with local memory and the crossbar.
  - The memory access is non uniform (NUMA) and the user does not control where threads are assigned.

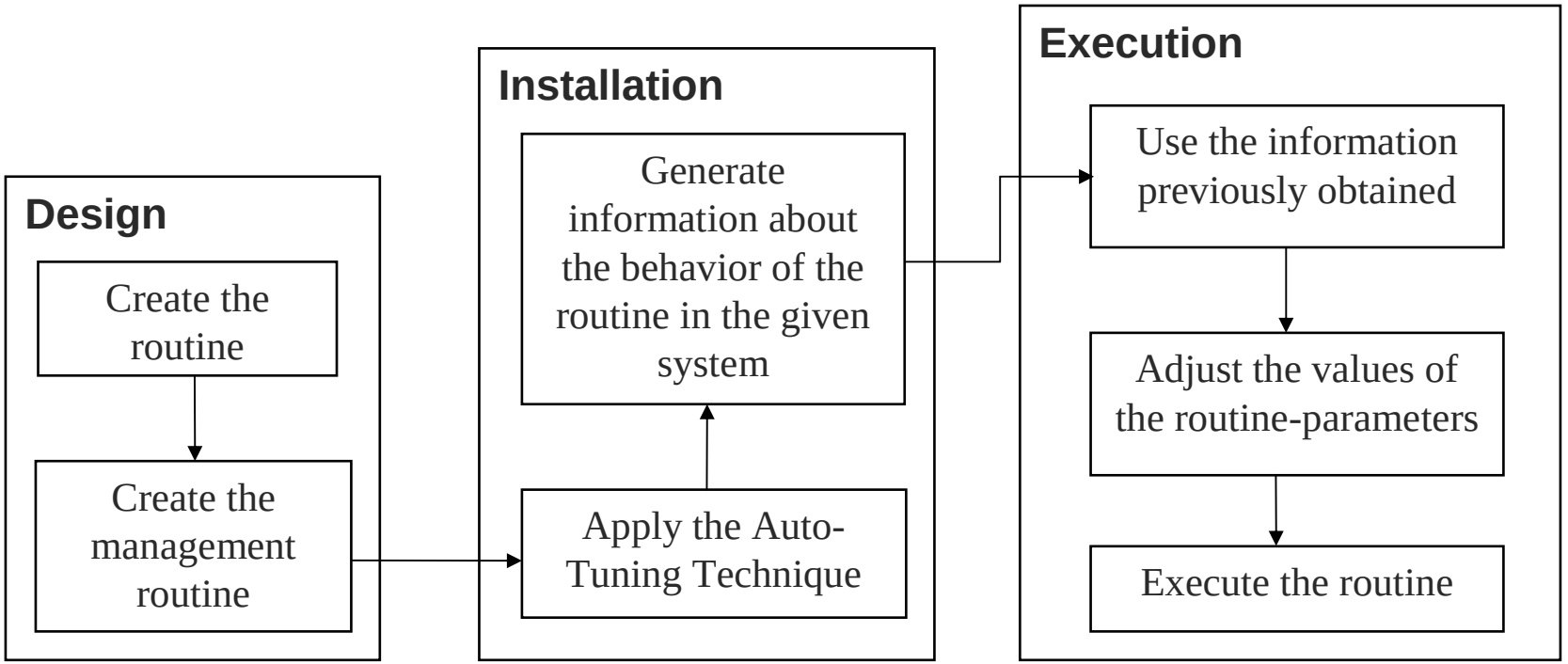


Two-Level Routines

- The loss of efficiency when the number of cores increases can be alleviated with multilevel parallelism.
- Two levels of parallelism (OpenMP+BLAS) are used:
  - It is necessary to apply some auto-tuning technique to select the number of threads to use at each level.
  - The selection can be made:
    - Through theoretical models of the execution time.
    - Using some installation methodology (in this work)

The Auto-Tuning Methodology

- Phases:



Application to Linear Algebra Subroutines

- Matrix multiplication routine (`gemm`) of BLAS:

Design:

- Syntax similar to BLAS routines:

```
dgemm2L(char transA, char transB, int m, int n, int k, double alpha,
        double *A, int lda, double *B, int ldb, double beta,double *C,
        int ldc, int thrOMP, int thrMKL)
```

Two new parameters to set the number of threads to use at each level of parallelism

- Nested parallelism scheme (OpenMP+MKL):

```
omp_set_nested(1);
mkl_set_dynamic(0);
omp_set_num_threads(nthomp);
mkl_set_num_threads(nthmkl);

#pragma omp parallel {
    obtain size and initial position of the submatrix of A
    call dgemm to multiply this submatrix by matrix B
}
```

Application to Linear Algebra Subroutines

- Matrix multiplication routine (`gemm`) of BLAS:

  Installation:

  - For each matrix size of the *installation set:*
    - Execute the routine in the system varying the number of threads at each level of parallelism using a combination not exceeding the maximum number of available cores.
    - Store the number of OpenMP and MKL threads with which the lowest execution time is obtained.
  - Result: the number of threads with which the lowest time is obtained for each problem size of the *installation set*.

Application to Linear Algebra Subroutines

- Matrix multiplication routine (`gemm`) of BLAS:

  Execution:

  For a specific problem size, the routine is executed with a number of OpenMP and MKL threads resulting from applying an interpolation process to the information stored during the installation phase.

  Example →

N	OMP (threads)	MKL (threads)
1000	2	8
...	...	...
2000	4	6
...	...	...
...	...	...
5000	6	4

```
N = 1500
OMP = (2+4)/2 = 3
MKL = (8+6)/2 = 7
```

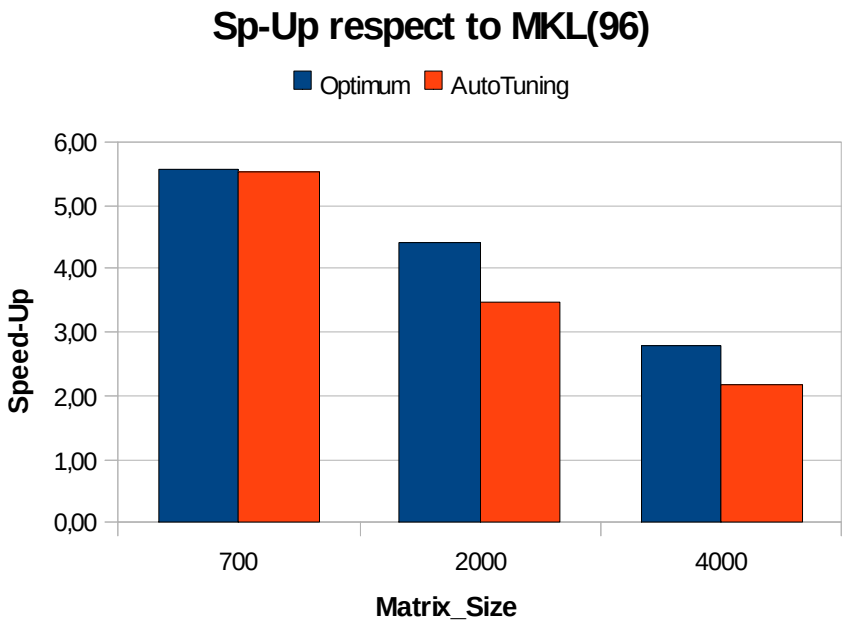
Application to Linear Algebra Subroutines

- Ben (*96 cores*):

  Installation Set = {500,1000,3000,5000}
  Validation Set = {700, 2000, 4000}

N	Optimum	Auto-Tuning	Sp-Up
700	0,0102 (*9x4*)	0,0103 (*10x4*)	0,99
2000	0,0749 (*10x6*)	0,0955 (*14x6*)	0,78
4000	0,4900 (*32x2*)	0,6287 (*22x4*)	0,78

In brackets: number of OpenMP+MKL threads with which the lowest execution time is obtained

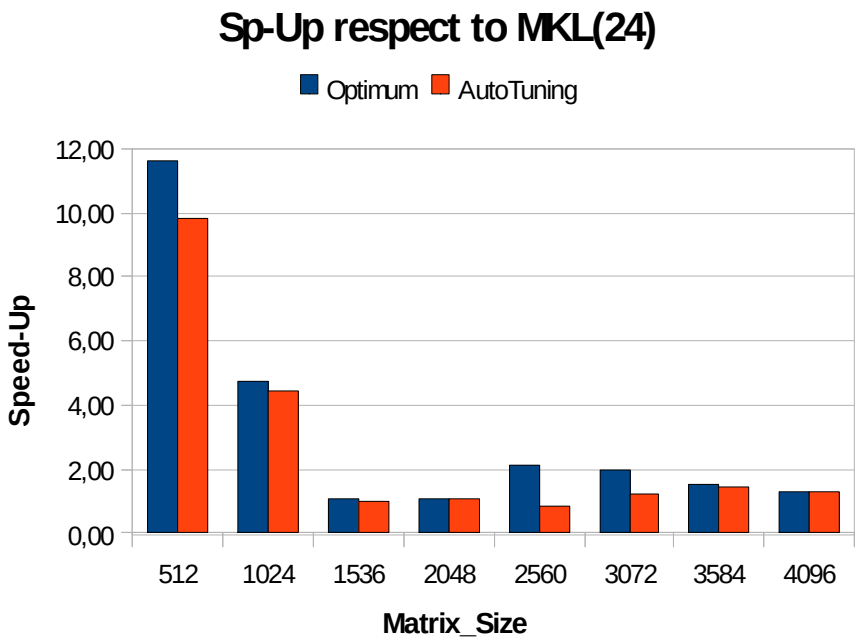


Application to Linear Algebra Subroutines

- Saturno (*24 cores*):

  Installation Set = {256, 768, 1280, 1792, 2304, 2816, 3328, 3840, 4352}
  Validation Set = {512, 1024, 1536, 2048, 2560, 3072, 3584, 4096}

	Optimum		AutoTuning	
N	OMP (threads)	MKL (threads)	OMP (threads)	MKL (threads)
512	1	16	1	14
1024	4	6	3	8
1536	4	6	6	4
2048	2	12	4	6
2560	7	3	6	4
3072	7	3	6	4
3584	3	8	4	6
4096	3	8	4	6



Application to Linear Algebra Subroutines

- Cholesky Decomposition Routine (`potrf`)
  - Used to solve a linear system of the type $AX=B$, with A symmetric definite positive.

Scheme used by LAPACK:

```
do 20 j=1, n, nb
    //Update and factorize the current diagonal block
    jb = min(nb, n-j+1)
    call dsyrk(...)
    call dpotf2(...)
    if (j+jb .le. n) then
        //Compute the current block column
        call dgemm(...)
        call dtrsm(...)
      endif
20 continue
```

Called using their corresponding multithreaded MKL implementation.

Replaced by the `dgemm` routine previously designed with 2 levels of parallelism (`dgemm2L`)

Application to Linear Algebra Subroutines

- Cholesky Decomposition Routine (`potrf`)

N	Optimum	AutoTuning	Speed-Up
512	0.003948 (9)	0.003793 (1x14)	1.04
1024	0.012877 (12)	0.011624 (3x8)	1.10
1536	0.024598 (24)	0.024420 (6x4)	1.00
2048	0.075525 (24)	0.076562 (4x6)	0.99
2560	0.109087 (24)	0.165639 (6x4)	0.66
3072	0.202955 (21)	0.237618 (6x4)	0.85
3584	0.279215 (21)	0.323004 (4x6)	0.86
4096	0.390708 (21)	0.383885 (4x6)	1.02

The results of applying the auto-tuning methodology are satisfactory.

The loss of performance for problem sizes {2560, 3072, 3584} is derived from the `dgemm2L` routine.

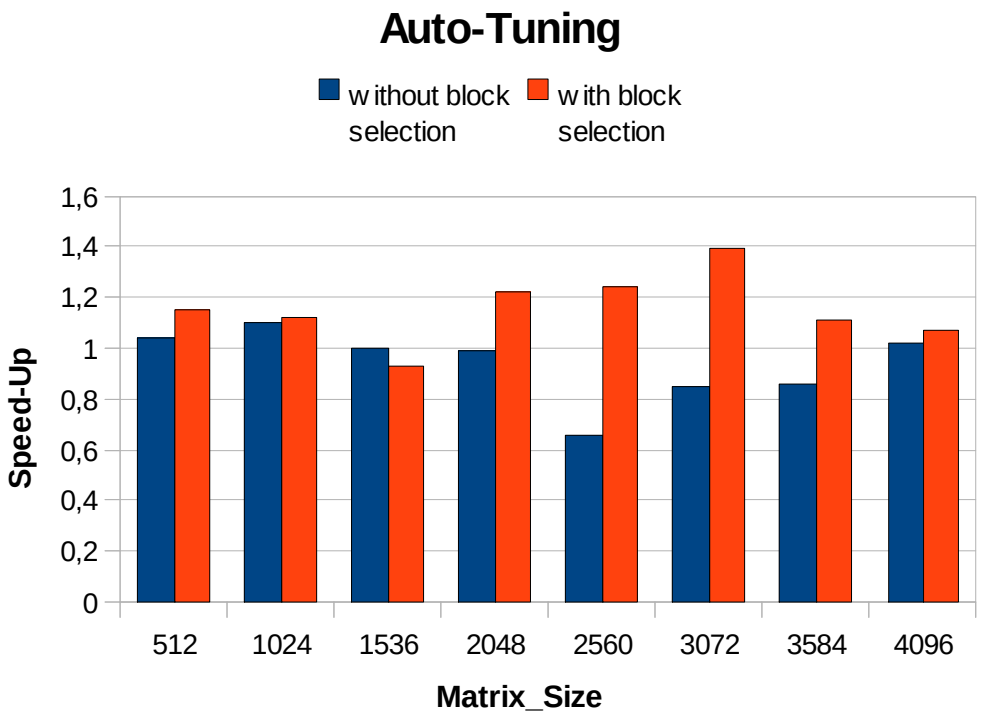
Application to Linear Algebra Subroutines

- Cholesky Decomposition Routine (`potrf`)
  - The Cholesky routine of LAPACK is computed by blocks.
  - Block size is internally determined by the ILAENV function. The value selected is not based on the number of threads → reduce the execution time by selecting the optimum block size.
  - Experiments have been done with block sizes power of 2 from 32 to 512 and using the same *installation set*.
  - By applying the auto-tuning methodology, lower execution times are obtained → improvement between 6% and 30%.
  - For small matrix sizes, the use of larger blocks is preferable, but for larger sizes, a smaller value for the block size is preferable.

Application to Linear Algebra Subroutines

- Cholesky Decomposition Routine (`potrf`)

	with_ILAENV	AutoTuning	
N	**Block-Size**	**Block-Size**	**Speed-Up**
512	32	128	1.15
1024	96	128	1.12
1536	192	64	0.93
2048	384	128	1.22
2560	384	64	1.24
3072	512	64	1.39
3584	512	256	1.11
4096	512	256	1.07



Conclusions

- The use of auto-tuning multithread routines in high-level routines is good for reducing the execution time in NUMA systems.
- An appropriate selection of the number of threads to use at each level of parallelism reduces the execution time, mainly for large matrices and when the number of cores increases.
- In algorithms by blocks, an appropriate selection of the block size together with an appropriate number of threads reduce the execution time even more.

Future Work

- Use the `dgemm2L` routine in other high-level linear algebra routines (LU, LDL^T) which are used in higher-level routines (such as `sysv`, `gesv`)
- Combine the empirical auto-tuning techniques with others based on theoretical models of the execution time (the LU decomposition is being studied)
- Apply this methodology to heterogeneus systems (clusters with GPU cards) and in routines of other linear algebra packages (PLASMA)

Thanks for your attention!