Empirical Auto-Tuning of Two-Level Parallel Linear Algebra Routines on Large cc-NUMA Systems

Jesús Cámara, Javier Cuenca, Domingo Giménez, Antonio M. Vidal
jcm23547@um.es, jcuenca@um.es, domingo@um.es, avidal@dsic.upv.es

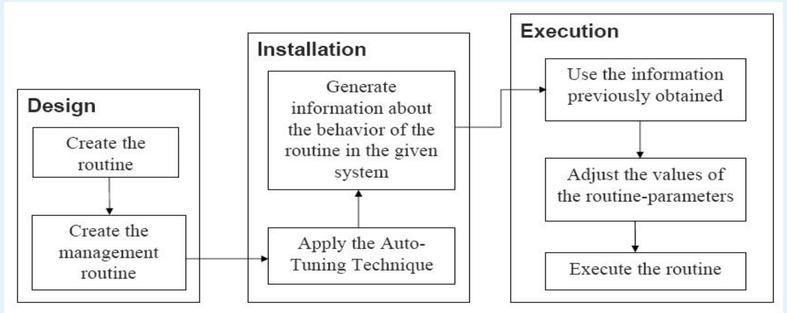
Motivation

Large scientific problems are solved in large computational systems using linear algebra libraries (BLAS, LAPACK). Multithread implementations of these libraries (MKL, PLASMA) are designed to obtain high performance in multicore and cc-NUMA systems. The efficient use of the different memory hierarchy levels in these systems is not an easy task and the performance of multithread implementations decreases when the number of cores increases:

To alleviate this problem, routines with multilevel parallelism can be developed by combining OpenMP and BLAS parallelism ⇒ apply some auto-tuning technique to select the appropriate number of threads to use at each level of parallelism to obtain the lowest execution time.

Auto-Tuning Methodology

Consist of three phases:



The methodology has been empirically applied. The goal is to select the most appropriate number of threads to use at each level of parallelism. The installation of the routine in the system is carried out by using an *installation set* consisting of a set of matrix sizes. The routine is executed in the system for each matrix size of the *installation set* by varying the number of threads at each level of parallelism and using a combination of OpenMP and MKL threads not exceeding the maximum number of available cores. As a result, the number of threads with which the lowest execution time is obtained is stored for each problem size of the *installation set*. At execution time and for a specific problem size, the routine is executed in the system with a number of OpenMP and MKL threads resulting from applying an interpolation process to the information stored during the installation phase.

Nested Parallelism Scheme

The methodology is applied to the BLAS-3 matrix multiplication routine (`dgemm`) with two-levels of parallelism (called `dgemm2L`):

```
omp_set_nested(1);
mkl_set_dynamic(0);
omp_set_num_threads(omp);
mkl_set_num_threads(mkl);
#pragma omp parallel
  obtain size and initial position of submatrix
  of A and call dgemm to multiply this submatrix
  by matrix B
```

The first level of parallelism is set with OpenMP and the second level is set by calling the multithread MKL implementation of the `dgemm` routine. The auto-tuning parameters (omp,mkl) are used to set the number of threads in each level.

Multicore Systems

Ben: cc-NUMA system with 128 cores, Intel-Itanium-2 Dual-Core processors (1.6 GHz), 1.5 TB of shared-memory. Located at the Supercomputing Centre of Murcia.

Saturno: NUMA system with 24 cores (4 hexa-core), Intel Xeon E7350 processors (1.87 GHz), 32 GB of shared-memory. Located at the Informatics Faculty at the University of Murcia.

Application to the Two Level Matrix Multiplication Routine

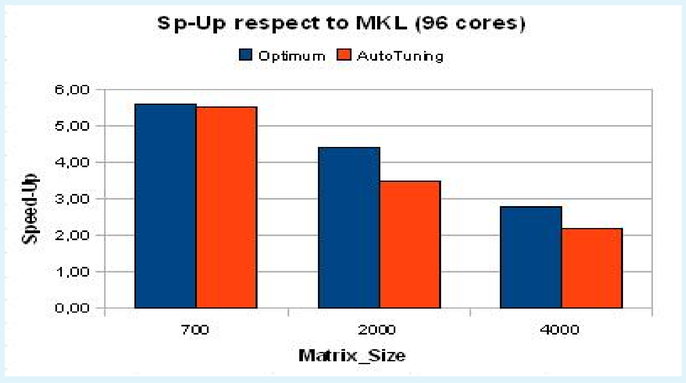
The auto-tuning methodology has been applied to the matrix multiplication routine because it is one of the most important computational kernels used in scientific computing. The *installation set* used to install the `dgemm2L` routine in the system is *{500, 1000, 3000, 5000}*. To determine how far from the optimum are the execution times obtained, the *validation set* = *{700, 2000, 4000}* is used.

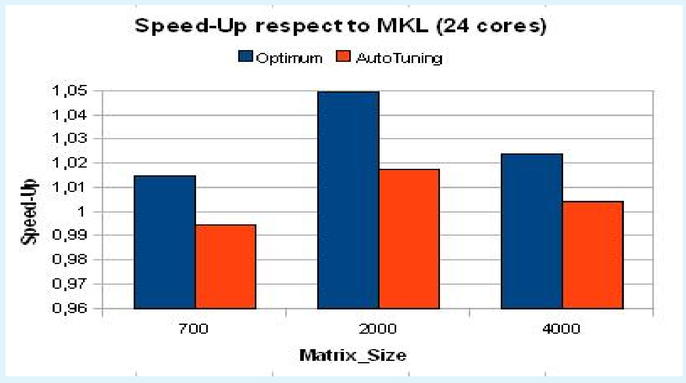
Ben

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

Saturno

N	Optimum	Auto-Tuning	Sp-Up
700	0.0098 (6x4)	0.0100 (8x3)	0.98
2000	0.2151 (6x4)	0.2218 (8x3)	0.97
4000	1.6076 (6x4)	1.6398 (8x3)	0.98



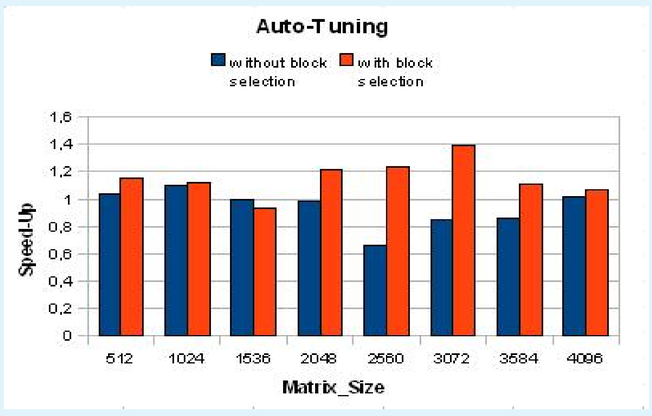


Tables show the execution times (in seconds) obtained in each system when the auto-tuning methodology is applied. Last column shows the speed-up achieved by the auto-tuning methodology respect to the optimum. In brackets it is shown the number of threads with which the lowest execution time is obtained. Charts show the speed-up achieved by the auto-tuning methodology respect to the use of only MKL parallelism with a number of cores equal to the available cores in the systems (96 in Ben and 24 in Saturno). The speed-up obtained is satisfactory and shows that the use of two levels of parallelism together with an auto-tuning technique is a good technique for reducing the execution time in multicore systems.

Application to High-Level Routines - Cholesky Factorization

The LAPACK Cholesky routine `dpotrf` is analyzed in order to study the improvement achieved by the auto-tuning methodology when it is included in high-level linear algebra routines. The `dgemm` routine internally used is replaced by the `dgemm2L` routine and its other internal routines are called using their corresponding multithreaded MKL implementation. All experiments have been carried out in Saturno and the *installation set* used is: {256, 768, 1280, 1792, 2304, 2816, 3328, 3840, 4352}. Left table shows the execution time (in seconds) obtained with the auto-tuning methodology and the speed-up achieved respect to the use of only MKL parallelism. Results are satisfactory and the total number of OpenMP and MKL threads used for each matrix size is similar.

	Selecting Threads			Selecting Threads+Block-Size		
N	Opt (LAPACK+MKL)	AutoTuning	Sp-Up	with-ILAENV	with-AutoTuning	Sp-Up
512	0.003948 (9)	0.003793 (1x14)	1.04	32	128	1.15
1024	0.012877 (12)	0.011624 (3x8)	1.10	96	128	1.12
1536	0.024598 (24)	0.024420 (6x4)	1.00	192	64	0.93
2048	0.075525 (24)	0.076562 (4x6)	0.99	384	128	1.22
2560	0.109087 (24)	0.165639 (6x4)	0.66	384	64	1.24
3072	0.202955 (21)	0.237618 (6x4)	0.85	512	64	1.39
3584	0.279215 (21)	0.323004 (4x6)	0.86	512	256	1.11
4096	0.390708 (21)	0.383885 (4x6)	1.02	512	256	1.07



The Cholesky routine is computed by blocks. Right table show the speed-up achieved when the block-size is also selected by the auto-tuning methodology respect to the value selected by the LAPACK ILAENV function. Experiments have been done with block sizes power of 2 from 32 to 512. The chart show the speed-up achieved by the auto-tuning methodology with this additional parameter respect to the use of the methodology without selecting the block size. The speed-up is higher and the improvement achieved is between 6% and 30%.

Funding

This work has been partially supported by the Conserjería de Educación de la Región de Murcia (Fundación Séneca, 08763/PI/08), and by the High-Performance Computing Network on Parallel Heterogeneus Architectures(CAPAP-H).