Characterization of Multicore Architectures using Task-Parallel ILU-type Preconditioned CG Solvers

José I. Aliaga[1] María Barreda[1] Enrique S. Quintana-Ortí[1]

We investigate the efficiency of state-of-the-art multicore processors using a multi-threaded task-parallel implementation of the Conjugate Gradient (CG) method, accelerated with an incomplete LU (ILU) preconditioner. Concretely, we analyze multicore architectures with distinct designs and market targets to compare their parallel performance and energy efficiency.

1 Introduction

The solution of sparse linear systems via iterative methods has been recently argued to be representative of the actual performance that is experienced by a large fraction of the scientific and engineering codes running on current supercomputers. This has led to the introduction of the HPCG benchmark[2] as a complement to the traditional LINPACK benchmark that ranks supercomputers twice per year in the Top500/Green500 lists.

Following this idea, in this work we investigate the efficiency of state-of-the-art multicore processors using our own task-parallel implementation of the CG method enhanced with an ILU-type preconditioner (hereafter, referred to as ILU-PCG). Our experimental analysis evaluates both the parallel performance and energy consumption of a variety of multicore architectures designed to deliver high performance and/or reduced energy consumption.

The rest of the abstract is organized as follows. In Section 2 we describe the ILU-PCG solver and how to extract the task-parallelism. In Section 3 we present the architectures included in the study. In Sectionand 4 we characterize the multicore architectures using the ILU-PCG solver. Finally, in Section 5 we offer some concluding remarks.

2 Task-parallel ILU-PCG solver

2.1 The iterative solver

In the evaluation we employ an ILU-PCG solver to tackle linear systems with sparse and symmetric positive definite (s.p.d.) matrix A. The solver computes a preconditioner M, via an ILU factorization of the coefficient matrix, and then solves the preconditioned linear system via a convenient variant of the CG method, which hopefully exhibits a fast convergence rate due to the effect of the preconditioner.

The most complex and challenging operations in the ILU-PCG benckhmark are the computation of the preconditioner M and its application. The remaining computations involve basic linear algebra operations such as the sparse matrix-vector product (SPMV),

[1]Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain, aliaga@uji.es, mvaya@uji.es, quintana@uji.es

[2]`http://www.hpcg-benchmark.org/`

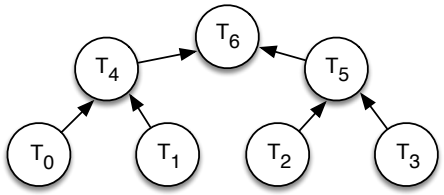

Figure 1: Dependency tree of the diagonal blocks. Task T_j is in charge of processing the diagonal block A_{jj}.

DOT (or inner) product, AXPY-like update, and the calculation of a vector 2-NORM (equivalent to a dot product). We will therefore focus the following analysis in the operations involving the preconditioner.

2.2 Task-Parallel PCG

The concurrency intrinsic to the calculation as well as the application of the preconditioner is considerably involved. Specifically, the parallelism of the process can be unveiled by means of nested dissection applied to the adjacency graph representing the non-zero connectivity of matrix A. By recursively splitting this graph, the result is a hierarchy of independent subgraphs, organized as dependency acyclic graph (DAG) with the shape of a binary tree. In the remaining operations of PCG, concurrency is extracted by dividing the operations into subtasks, and mapping the data into the leaves of the DAG. For example, it is straight-forward to partition the output vector from SPMV (or the AXPYs) into several subvectors, which can be then computed as independent tasks. The DOT and vector 2-NORM are reduction-type operations, also parallelizable, but impose a global synchronization/communication point to the procedure.

Figure 1 illustrates the dependency tree for the factorization of the diagonal blocks. The edges of the DAG define the dependencies between the diagonal blocks (tasks); that is, the order in which these blocks of the matrix have to be processed. Luckily, the leaf tasks of the DAG in general comprise a significant part of the computational cost of the process.

In summary, by recursively applying the same idea, we can explicitly unveil an increasing amount of task-level parallelism during the factorization that computes the preconditioner M as well as the triangular solves involved in its application. The DAGs associated with the first stage (computation of the preconditioner) and the triangular solves with the lower triangular factor present the form of a tree with bottom-up dependencies, from the leaves to the root; on the other hand, the triangular solves with the upper triangular factor share the structure of the DAG but the dependencies are reversed, pointing down from the root to the leaves.

We note that the recursive decomposition of the graph into further levels multiplies the concurrency exponentially. However, there exists a balance between the number of levels, and consequently independent tasks, and the convergence rate of the procedure. Concretely, this recursive process introduces additional numerical levels in the computation of the preconditioner. Thus, different DAGs are associated with distinct preconditioners, which can be expected to feature close numerical properties.

In our particular parallel implementation of the PCG solver, task-parallelism is ex-

Architecture	SANDY	ODROID(A15)	JUNO(A57)	HASWELL	XEON PHI
PROCESOR NUMBER	E5-2620	ARMv7 rev 3 (v7l)	AArch64 rev 0	E5-2603v3	5110P
#SOCKETS	2	1	1	2	1
#CORES	12	4	2	12	60
BASE FREQUENCY	2.0 GHz	2.0 GHz	1.1 GHZ	1.6 GHz	1.053
CACHE	15 MB	2 MB	2 MB	15 MB	30 MB
TDP	95 W	15 W	30 W	85 W	225 W
VOLTAGE RANGE	0.60 V-1.35 V	0.91 V-1.32 V	0.81 V-1.00 V	0.65 V-1.30 V	–
MEMORY	32 GB	2 GB	8 GB	32 GB	8 GB
MAX. MEMORY BANDWIDTH	42.6 GB/s	14.9 GB/s	13.2 GB/s	51 GB/s	320 GB/s

Table 1: Hardware specifications of the platforms.

Architecture	SANDY	ODROID (A15)	JUNO (A57)	HASWELL	XEON PHI
GCC	4.4.6	4.8.2	4.9.1	4.4.7	5.1.0
OMPSS	16.06	16.06.1	16.06.1	16.06.1	16.06
MERCURIUM	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0
NANOX	0.12a	0.10.1	0.10.1	0.10.1	0.12a
METIS	5.0.2	5.0.2	5.0.2	5.0.2	5.0.2
POWER/ENERGY MEASUREMENTS	RAPL	PMLIB	PMLIB	RAPL	PMLIB
FREQUENCY CHANGES	CPUfreq	CPUfreq	CPUfreq	CPUfreq	–

Table 2: Software specifications of the platforms.

ploited using a multi-threaded code that relies on the OmpSs programming model[3] [1].

3 Target Multicore Architectures and General Setup

For the study, we selected three different types of multicore architectures comprising two general-purpose processors from Intel, two low-power systems from ARM, and the Intel Xeon Phi. This collection is representative of todays' multicore technology. Table 1 offers some information about hardware in each platform; and the software employed in the platforms are described in Table 2.

All the experiments in next sections employed IEEE754 real double-precision arithmetic. For the analysis, we employed a large-scale linear system corresponding to the Laplacian equation $-\Delta u = f$ in a $3D$ unit cube $\Omega = [0, 1]^3$ with Dirichlet boundary conditions, $u = g$ on $\partial\Omega$, and a discretization that resulted in a SPD system, with instances of different size. The problem size is the largest that fits in the main memory of each platform.

4 Characterizing Multicore Architectures using ILU-PCG

The following experimental evaluation comprises three "dimensions", two of them architectural (number of cores/threads and operation frequency) and one corresponding to software (number of leaves for ILU-type preconditioner task dependency tree). Due to the large number of tests and results, we organize the presentation of the performance analysis of the architecture into the following sequence of steps:

[3] `https://pm.bsc.es/ompss`

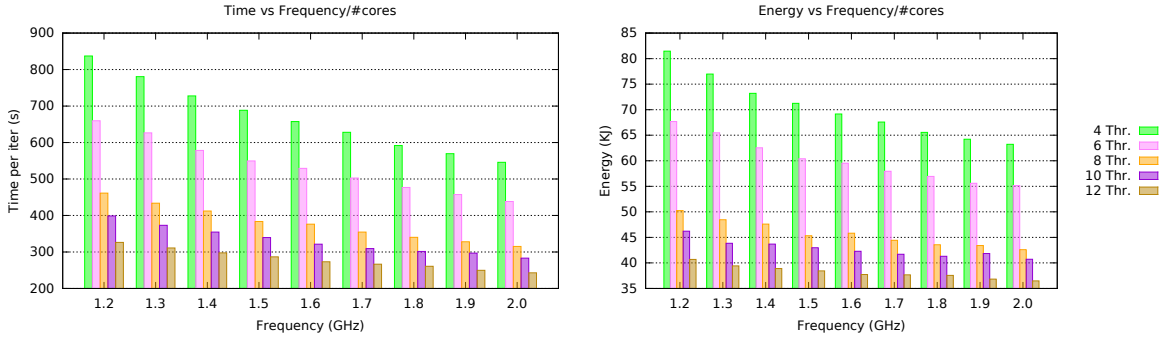

Figure 2: Time and energy consumption for the execution of ILU-PCG in SANDY.

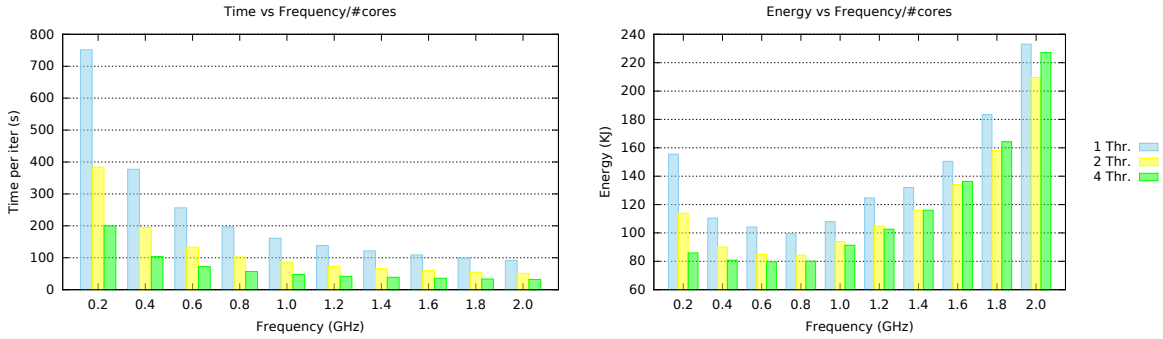

Figure 3: Time and energy consumption for the execution of ILU-PCG in ODROID.

1. Evaluation of the parallel ILU-PCG solver for a range of representative frequencies and number of threads (thread-level parallelism), using 1, 2, 4,. . . , 64 leaves.

2. Selection of the optimal number of leaves for each level of thread-parallelism.

3. Evaluation of the impact of frequency on the ILU-PCG solver.

4. Selection of the optimal frequency for each number of threads.

5. Evaluation of the impact of the thread-level parallelism on the ILU-PCG solver.

This analysis is then repeated from the perspective of energy efficiency, taking as a basis the performance evaluation to justify some of the results for this second metric.

In order to avoid an exhaustive list of figures and results, we next summarize them using a few plots that illustrate the interplay between frequency/thread concurrency and performance/energy consumption. In particular, Figures 2–6 report absolute values for the last two metrics against processor frequency and number of threads. The third one (number of leaves), depends on the software, and is set for all these experiments to the optimal value. To allow an easier visualization of the differences, for those architectures with a large number of cores, we skip the results obtained with 1 and 2 cores. In any case, these configurations always offered worse performance and energy efficiency than those using a higher level of thread concurrency.

A collection of general remarks can be extracted from this experimental evaluation that emphasize the differences between the performance-oriented architectures (Intel Xeon) and the low-power processors (ARM):

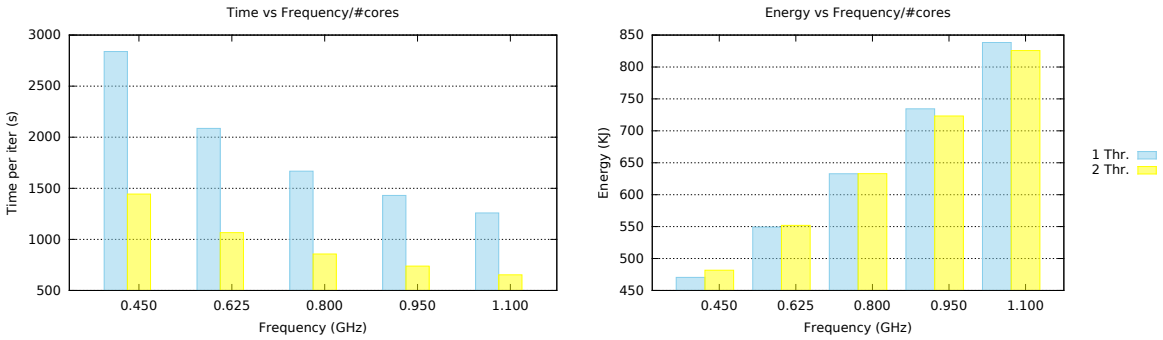

Figure 4: Time and energy consumption for the execution of ILU-PCG in JUNO.

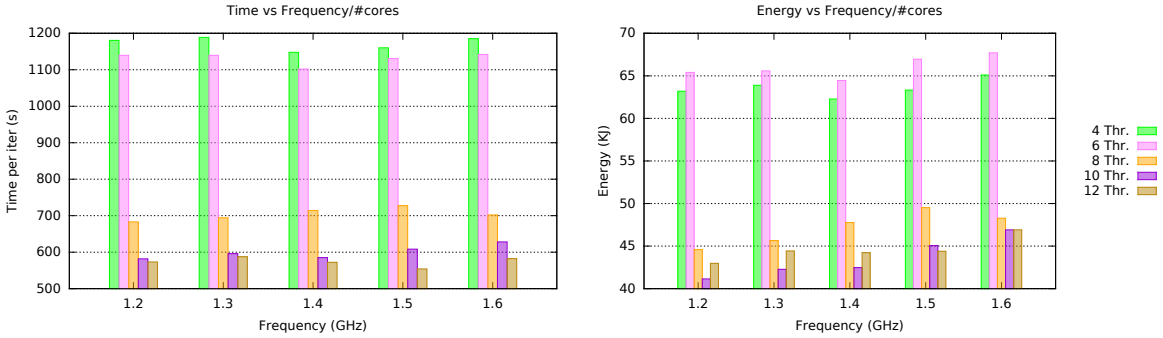

Figure 5: Time and energy consumption for the execution of ILU-PCG in HASWELL.

Performance:

1. The optimal number of leaves is mostly determined by the problem size: a larger dimension can accommodate additional levels of task-parallelism without incurring into a costly overhead. In contrast, the number of leaves is basically independent of the architeture class (performance-oriented versus low-power), frequency, and number of threads. For the small and large problem instances, the optimal number of leaves are, respectively, 8 and 32.
2. The execution time in general benefits from operating at a higher frequency and/or using a larger number of cores. However, the differences may be small when the memory bandwidth is saturated as the results for the low-power architecture demonstrate.

Energy consumption:

1. The optimal numbers of leaves match those obtained when the figure-of-merit is performance. The same remarks apply when the target metric is energy consumption.
2. The optimal frequency is the highest one for the Intel performance-oriented architectures. In contrast, the ARM low-power processors benefit from a more reduced frequency level. The reason for this different behaviour is twofold, and can be used to further distinguish the behaviour of the two types of systems:
   a) The performance-oriented architectures exhibit a considerable static power rate so that increasing the execution time is very costly in terms of energy consumption. The low-power processors do not suffer from this drawback.

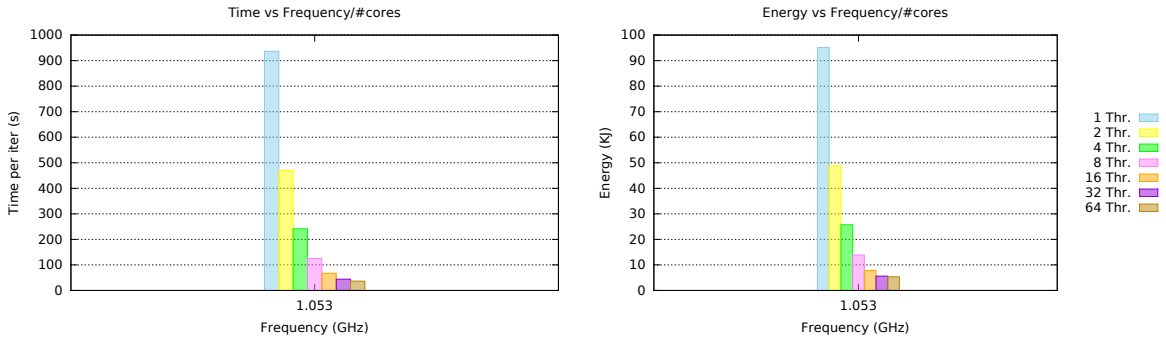

Figure 6: Time and energy consumption for the execution of ILU-PCG in XEON PHI.

b) The low-power processors tend to saturate the memory bandwidth rapidly as the frequency is raised, yielding a negligible improvement of execution time for a linear increase in the power dissipation rate. The consequence is a worse energy efficiency.

3. From the perspective of scalability, adding more cores is beneficial unless the memory bandwidth is saturated. Once that threshold is surpassed, the increase in the dissipation rate directly translates into higher energy costs.

5 Conclusions

We have analyzed the computational performance and energy efficiency of servers equipped with the state-of-the-art general-purpose multicore processors as well as accelerators like the Intel Xeon Phi. Following the introduction of the HPCG benchmark, we adopted ILU-PCG to test performance and energy efficiency of multicore platforms, observing different behaviours for performance-oriented and low-power processors, especially when the figure of merit is energy efficiency.

References

[1] J. I. Aliaga, R. M. Badia, M. Barreda, M. Bollhöfer, and E. S. Quintana-Ortí. Leveraging task-parallelism with OmpSs in ILUPACK's preconditioned CG method. In *26th Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD 2014)*, pages 262–269, 2014.