Parallel Computing 000 (2017) 1–18

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Revisiting conventional task schedulers to exploit asymmetry in multi-core architectures for dense linear algebra operations

Luis Costero[a], Francisco D. Igual[a,*], Katzalin Olcoz[a], Sandra Catalán[b], Rafael Rodríguez-Sánchez[b], Enrique S. Quintana-Ortí[b]

[a] Depto. de Arquitectura de Computadores y Automática, Universidad Complutense de Madrid, Madrid, Spain
[b] Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón, Spain

ARTICLE INFO

Article history:
Received 14 October 2016
Revised 9 May 2017
Accepted 1 June 2017
Available online xxx

Keywords:
Linear algebra
Task parallelism
Runtime task schedulers
Asymmetric architectures

ABSTRACT

Dealing with asymmetry in the architecture opens a plethora of questions related with the performance- and energy-efficient scheduling of task-parallel applications. While there exist early attempts to tackle this problem, for example via *ad-hoc* strategies embedded in a runtime framework, in this paper we take a different path, which consists in addressing the asymmetry at the library-level by developing a few asymmetry-aware fundamental kernels. The appealing consequence is that the architecture heterogeneity remains then hidden from the task scheduler.

In order to illustrate the advantage of our approach, we employ two well-known matrix factorizations, key to the solution of dense linear systems of equations. From the perspective of the architecture, we consider two low-power processors, one of them equipped with ARM big.LITTLE technology; furthermore, we include in the study a different scenario, in which the asymmetry arises when the cores of an Intel Xeon server operate at two distinct frequencies. For the specific domain of dense linear algebra, we show that dealing with asymmetry at the library-level is not only possible but delivers higher performance than a naive approach based on an asymmetry-oblivious scheduler. Furthermore, this solution is also competitive in terms of performance compared with an *ad-hoc* asymmetry-aware scheduler furnished with sophisticated scheduling techniques.

1. Introduction

The end of Dennard scaling [1] has promoted heterogeneous systems into a mainstream approach to leverage the steady growth of transistors on chip dictated by Moore's law [2,3]. *Asymmetric multicore processors* (AMPs) are a particular class of heterogeneous architectures that combine two (or more) distinct types of cores, but differ from multicore servers equipped with hardware accelerators in that the AMP cores share the same instruction set architecture (single-ISA) and a strongly coupled memory subsystem. A representative example of these architectures are ARM-based AMPs, consisting of a cluster of high performance (BIG) cores plus a cluster of low-power (LITTLE) cores. These architectures can, in theory, deliver much higher performance for the same power budget and adapt better to heterogeneous application ecosystems [4].

* Corresponding author.
E-mail addresses: lcostero@ucm.es (L. Costero), figual@ucm.es (F.D. Igual), katzalin@ucm.es (K. Olcoz), catalans@uji.es (S. Catalán), rarodrig@uji.es (R. Rodríguez-Sánchez), quintana@ucm.es (E.S. Quintana-Ortí).

http://dx.doi.org/10.1016/j.parco.2017.06.002
0167-8191/

A different scenario leading to asymmetric configurations arises when a conventional system equipped with a symmetric multicore architecture, e.g. an Intel Xeon multisocket server, has to regulate its sockets/cores to operate at two (or more) voltage-frequency scaling (VFS) levels, during a certain period, in order to meet a certain power budget.

Task parallelism has been reported as an efficient means to leverage the considerable number of cores in current processors. Several efforts, pioneered by Cilk [5], aim to ease the development and improve the performance of task-parallel programs by tackling task scheduling from inside a *runtime* (framework). The benefits of this approach for complex dense linear algebra (DLA) operations have been demonstrated, among others, by OmpSs [6], StarPU [7], PLASMA/MAGMA [8,9], and `libflame` [10]. In general, the runtimes underlying these tools decompose DLA routines into a collection of numerical kernels (or tasks), and then take into account the dependencies between the tasks in order to correctly issue their execution to the system cores. The tasks are therefore the "indivisible" scheduling unit while the cores constitute the basic computational resources.

Task-based programming models and asymmetric architectures (specifically, low-power ones such as the ARM big.LITTLE processors) have recently attracted the interest of the scientific community to tackle the programmability and energy consumption challenges, respectively, on the road towards Exascale [11,12]. Two questions arise when combining both approaches: first, will these architectures require a complete redesign of existing runtime task schedulers? Second, how will conventional (asymmetry-oblivious) runtimes behave when operating under asymmetric architectures?

In this paper we address both questions by introducing an innovative and efficient approach to execute task-parallel DLA routines on AMPs via *conventional asymmetry-oblivious schedulers*. Conceptually, our solution aggregates the cores of the AMP into a number of *symmetric virtual cores* (VCs), which become the only basic computational resources that are visible to the runtime scheduler. In addition, a specialized implementation of each type of task, embedded into an asymmetry-aware DLA library, partitions each numerical kernel into a series of fine-grain computations, which are efficiently executed by the asymmetric aggregation of cores of a single VC. Our work thus makes the following specific contributions:

- We target the Cholesky and LU factorizations [13], two complex operations for the solution of linear systems that are representative of many other DLA computations. Therefore, we are confident that our approach and results carry over to a significant fraction of the DLA functionality covered by LAPACK (*Linear Algebra PACKage*) [14].
- For these two DLA operations, we describe how to leverage the asymmetry-oblivious task-parallel runtime scheduler in OmpSs, in combination with a data-parallel instance of the BLAS-3 (*basic linear algebra subprograms*) [15] in the BLIS library specifically designed for AMPs [16,17].
- We provide practical evidence that, compared with an *ad-hoc* asymmetry-conscious scheduler recently developed for OmpSs [18], our solution yields higher performance for the multi-threaded execution of the LU and Cholesky factorizations on four different asymmetric configurations, including state-of-the-art instances of ARM big.LITTLE and VFS-configured Intel Xeon systems.
- In conclusion, compared with previous work [16,18], this paper demonstrates that, for the particular domain of DLA, it is possible to hide the difficulties intrinsic to dealing with an asymmetric architecture (e.g., workload balancing for performance, energy-aware mapping of tasks to cores, and criticality-aware scheduling) inside an asymmetry-aware implementation of the BLAS-3. In consequence, our solution can refactor any conventional (asymmetry-agnostic) scheduler to exploit the task parallelism present in complex DLA operations.

This paper is an extension of the work presented in [19]. In particular, the experimental analysis of the LU factorization with partial pivoting, as well as the study with ARM 64-bit AMPs and VFS-asymmetric scenarios were not analyzed in [19].

The rest of the paper is structured as follows. Section 2 presents the target asymmetric configurations, including two ARM big.LITTLE AMPs and the Intel Xeon architecture. Section 3 reviews the state-of-the-art in runtime-based task scheduling and DLA libraries for (heterogeneous and) asymmetric architectures. Section 4 introduces the approach to reuse conventional runtime task schedulers on AMPs by relying on an asymmetric DLA library. Section 5 reports the performance results attained by the proposed approach on different architectures for both the Cholesky and LU with partial pivoting factorizations; more specifically, Section 5.3 dives into performance details for a given architecture (Exynos) and operation (Cholesky factorization). Finally, Section 6 closes the paper with the concluding remarks.

2. Asymmetric configurations and software support

2.1. Target asymmetric configurations

The architectures employed to evaluate our solution can be divided in two broad groups: *physically-asymmetric architectures*, in which the big and LITTLE clusters differ in the microarchitecture and (possibly) in the processor frequency; and *virtually-asymmetric architectures*, where the two clusters differ only in their frequency. Table 1 summarizes the main characteristics of the three target architectures that yield the four asymmetric configurations for the experimentation.

In the first group, Exynos is an ODROID-XU3 board comprising a 32-bit Samsung Exynos 5422 SoC with an ARM Cortex-A15 quad-core processing cluster (1.3 GHz) and a Cortex-A7 quad-core processing cluster (1.3 GHz). Each core has a private 32+32-Kbyte L1 (instruction+data) cache. The four A15 cores share a 2-Mbyte L2 cache, while the four A7 cores share a smaller 512-Kbyte L2 cache.

Please cite this article as: L. Costero et al., Revisiting conventional task schedulers to exploit asymmetry in multi-core architectures for dense linear algebra operations, Parallel Computing (2017), http://dx.doi.org/10.1016/j.parco.2017.06.002