Task Dependences Management Hardware Acceleration for Task-based Dataflow Programming models

Xubin Tan, Carlos Álvarez-Martínez, Daniel Jiménez-González, Eduard Ayguadé, Mateo Valero
Universitat Politècnica de Catalunya, Barcelona Supercomputing Center, Barcelona, Spain
{xubin.tan, eduard, maeto.valero}@bsc.es, {djimenez, calvarez}@ac.upc.edu

Abstract- *Task-based programming models have gained a lot of attention for being able to explore high parallelism over multicore and manycore, while hiding the difficulties of parallel programming. For applications with moderate size tasks, performance gains are assured by using these programming models. While for more parallelism by using smaller and more tasks, the performance degrades as a result of runtime overheads. To speed up the runtime, we present a hardware accelerator, Picos Hardware to accelerate task dependence management and scheduling. In this work, we show the performance of the first Picos Hardware prototype realized in a Zynq 7000 All-Programmable SoC by using real benchmarks. Results show that our hardware support greatly outperforms the software-only implementation currentlyavailable in the runtime system for fine-grained tasks.*

I. INTRODUCTION

Parallel computing offers the possibility to scale up the performance over the number of processors. At the same time, it exposes significant challenges for programmers to adapt themselves from sequential to parallel programming. Task-based programming models are quickly developed to target these challenges. For example, Google's MapReduce, Intel's TBB, Open MP 4.0, StarSs and OmpSs programming model [1]. In OmpSs, programmers can gain performance by simply annotating tasks in the source code with directives (input, output, inout) to hint their data dependences. And the remaining actions as task creation, dependence management/dependence graph management and task scheduling are managed by the Nanos++ Runtime system (RTS).

OmpSs is able to expose high parallelism from

moderate tasks, with both regular and irregular dependence patterns and is fairly easy to use. However, for fine-grained tasks, the runtime overheads (especially dependence management and task scheduling) are too high to scale.

To speedup the runtime and extend the usage of OmpSs to fine-grained tasks, we present a hardware accelerator, Picos Hardware. It accepts general information from master threads as task identification, number of dependences, memory addresses and directions of dependences, and schedules ready tasks to worker threads. In this work, we show a brief description of Picos and its hardware costs, and discuss some challenges during the development, and finally results of the first Picos prototype [2] realized in a Zynq 7000 All-Programmable SoC [3] by using real benchmarks.

II. METHODOLOGY AND CHALLENGES

A. *Experimental Setup*

First, OmpSs applications are executed in sequential and parallel up to 24 threads in a shared memory machine. It has 2 sockets, each socket is a Xeon E5-2630L with 6 cores with dynamic frequency up to 2.0GHz.

Second, execution time of the same applications of Picos full system are obtained in Zedboard (Zynq 7000 SoC) by using traces. The traces are obtained through instrumenting the sequential execution of OmpSs applications. It includes two main parts: the first part includes task creation/execution time in cycles required to simulate the task creation and task execution processes in ARM processor; the second part includes task and dependence information necessary for dependence management and task scheduling.

Finally, the speedups of OmpSs applications shown in this work are obtained against the sequential execution time.

B. *Picos Full System*

ion of the current embedded system integrated with the first Picos prototype. The Programmable Logic part uses a 80MHZ global clock, and a 64bits AXI Timer synchronized with the same clock as the global timer. The ARM processor runs a bare-metal Operating System, and the workers inside are simulating threads.

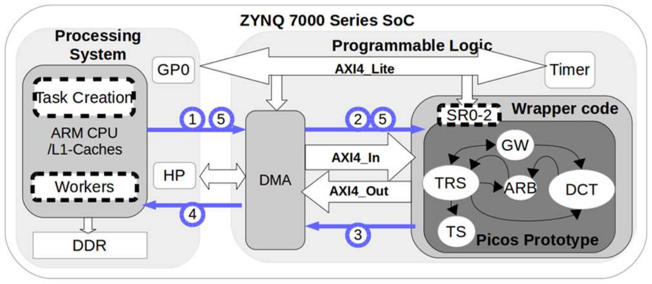

Fig. 1. The Picos Full System employs a close-loop process. Each task is created and sent to Picos for dependence analysis (1, 2). Each ready task is retrieved from Picos to the ARM core for execution in the workers (3, 4). Finally each finished task is sent back to notify Picos (5) to carry on the process until the last task.

Each message between Picos and ARM core carries one task at once and the communication latency for sending or retrieving each task via DMA takes around 200 to 300 cycles for each message.

C. Picos Prototype

Picos prototype has five functional units: Gateway (GW), Task Reserve Station (TRS), Dependence Chain Tracker (DCT), Task Scheduling (TS) and Arbiter (ARB).

GW reads new/finish task information from workers to Picos prototype.

TRS is the major task management unit. It stores in-flight tasks, tracks the readiness of new tasks and manages the deletion of finished tasks.

DCT is the major dependence management unit. It performs address matching of new dependence against the addresses of those arrived earlier, to track data dependences, and also save and control all its live versions.

TS schedules ready tasks notified by TRS to idle workers.

ARB manages communications between TRS and DCT.

The first prototype uses about 5.8% Look-Up Tables, 1.2% Flip-Flops and 17% BRAMs in XC7Z020 [3].

D. Challenges

We encounter several big challenges during the development of the first Picos prototype. Firstly, the balance between speed and hardware cost of TRS and DCT. Since each task can have multiple dependences, this stresses the dependence management unit multiple times more than the task dependence unit. Secondly, the system stalls if new dependences cannot be processed due to the memory capacity and entry conflicts. Thirdly, the communication latency between Picos prototype and the ARM processor.

III. RESULT EVALUATION

We show the speedup (y-axis) of Cholesky, SparseLu (four different block sizes) [4] obtained by Picos Full-system, Perfect Simulator and Nanos++ RTS, with up to 24 threads in Fig. 2.. Results of Perfect Simulator shows the critical-path roofline speedup.

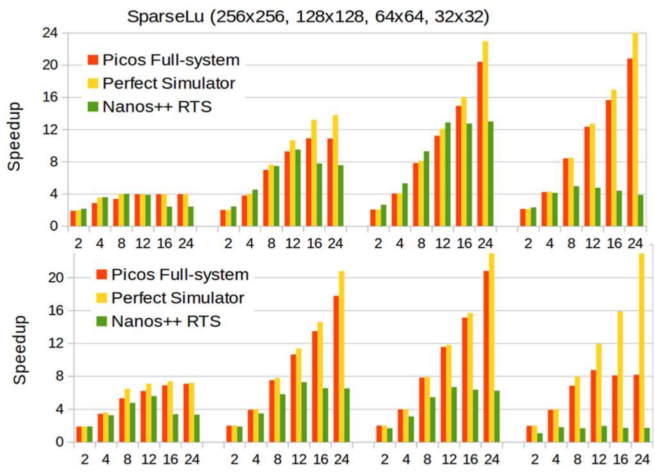

Fig. 2. Speedup of OmpSs applications with 2 to 24 threads

As can be seen, firstly, for each benchmark with a fixed block size, Nanos++ RTS scales up to 12 workers maximum while the Picos prototype continues to scale to 24 workers. For example, for SparseLu and Cholesky in with block size 64, the Picos prototype achieves over 20x with 24 workers while Nanos++ RTS achieves 13x and 7x respectively.

Secondly, for both benchmarks, Nanos++ RTS starts to degrade rapidly after some point s while the Picos prototype keeps on advancing or remains stable as the block size decreases. For Cholesky, although the performance of both Picos prototype and Nanos++ RTS degrade for block size 32, the latter one has a much worse degradation. The reason for the Picos prototype degradation here is that it only uses one TRS and DCT, which is unable to unfold such a high parallelism from Cholesky with block size 32. However, with more module instances Picos Hardware should be able to obtain higher speedup and fill this gap [5].

IV. CONCLUSIONS

In this paper we show a brief description of Picos, as a RTS hardware support to speedup the task and dependence management for task-based dataflow programming models like Open MP 4.0 and OmpSs. The presented implementation has been in a Zynq 7000 All-programmable SoC

Platform. Results of real benchmarks show that the prototype greatly outperforms the existing OmpSs software-only implementation (Nanos++) and as the task granularity decreases, the prototype continues to scale after Nanos++ RTS starts to degrade. More importantly, with a larger design with multiple task and dependence management units upcoming, Picos Hardware could be able to exploit a larger magnitude of parallelism in the applications with very fine granularity, that software alternatives cannot achieve.

ACKNOWLEDGMENT

This work is supported by the Programa Severo Ochoa (SEV-2015-0493) through the TIN2015-65316-P project, the contracts 2014-SGR-1051 and 2014-SGR-1272, and the European Research Council RoMoL Grant Agreement number 321253. We also thank the Xilinx University Program. This work has one accepted paper [2].

REFERENCES

[1] A. Duran, E. Ayguade, R. M. Badia, J. Labarta, L. Martinell, X. Mar-torell, and J. Planas, "Ompss: A proposal for programming heteroge- neous multi-core architectures," Parallel Processing Letters, 2011.

[2] X. Tan, J. Bosch, D. Jiminez-Gonzalez , C. Alvarez-Martinez, E. Ayguade and M. Valero. "Performance Analysis of a Hardware Accelerator of Dependence Management for Task-based Dataflow Programming models". Accepted to the 2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS).

[3] XILINX, "Zynq-7000, etc.." [online], 2015. http://www.xilinx.com/support/documentation/user guides/ug585-Zynq-7000-TRM.pdf.

[4] B. S. Center, "Bsc application repository(bar)." [online], 2014. https//pm.bsc.es/projects/bar/wiki/Applications.

[5] F. Yazdanpanah, C. Alvarez, D. Jimenez-Gonzalez, R. M. Badia, M. Valero, "Picos: A hardware runtime architecture support for ompss," Future Generation Computer Systems(FGCS), 2015.