UNIVERSITY OF AMSTERDAM

On the Exploration of the DRISC Architecture

Q. Yang

Summary

The miniaturization of transistors that drives Moore's law enables insistent investment of logic on chip. This has already shifted processor design from building monolithic uniprocessors to integrating more and more cores with regard to sustainable performance efficiency and energy consumption. Multi-core processors with ever-increasing core count will definitely prosper in the computing landscape, despite the inevitable disagreements on how to design them and the challenges to be conquered in those designs once deployed.

The Dynamically-scheduled Reduced Instruction Set Computer (DRISC) architecture, advanced on the eve of multi-core processors, has been used to explore how Instruction-Level Parallelism (ILP) can be traded off against and how multithreading could be provisioned with simpler circuits to break the roof imposed by hardware complexity and power dissipation. Its simplicity and elegance remains attractive and promising as a basis of multi-core or many-core chips for high system throughput using multithreading and in-order issue.

To promote DRISC for general-purpose computing, the Computer System Architecture group of the University of Amsterdam directed its efforts into building a scalable many-core system based on DRISC. This has been achieved in collaboration with other technical partners and has made a contribution to the scalability and generality in the exploitation of concurrency in both software and hardware. However, although an individual DRISC core can perform remarkably well with sufficient concurrency, as a part of the many-core system, it suffers from the sharing of on-chip interconnections and off-chip pins in the context of massive parallelism. The resultant bandwidth problem is shared by other many-core research designs and is detrimental to system scalability. This thesis extends this prior work by investigating system performance and by broadening the application of this processor design.

First we explore how performance is saturated or degraded by either increasing the number of cores or hardware threads per core. We monitor on-chip traffic and notice how coherence transactions scale with on-chip concurrency and how this affects system throughput. To mitigate such hardware coherence overheads, we explore both software and hardware optimizations, and finally reintroduce the philosophy of write combining in the context of the DRISC processor and the on-chip distributed cache structures. We demonstrate how such a strategy has significant advantages in cost efficiency over that of larger caches and is more effective than refactoring applications to minimize traffic. It also manifests an ability to moderate pin-bandwidth pressure without increasing the number of memory controllers. All achievements are credited to the capability of reducing the network payloads

or alleviating the concurrent or peak stress. Although not applicable universally we show there is no negative impact on performance.

Domain-specific computing may also exploit multiple cores to accelerate jobs. Real-time processing is a good example of this but perhaps with modest numbers of cores. To always guarantee strict timing bounds, dedicated scheduling is often applied but this leads to low processor efficiency and thus wastes energy. Considering that a 90% utilization rate in DRISC cores is not unusual in a small scale multi-core system, we would like to exploit DRISC in real-time applications and share the benefits of data-flow scheduled multiple threads over multiple cores. The question explored here is whether we can enhance specific single or multi-thread real-time task performance while still meeting timing requirement and maintaining a high efficiency, or even under a general workload.

The results of conventional all-for-one and core-reservation paradigms demonstrate the real-time competence of the existing DRISC in the small-scale multi-core configurations using both aperiodic and periodic benchmarks. Nevertheless only through time-sharing can all tasks benefit from maximal computation power but with the premise of real-time awareness and assurance. In order to achieve that, we introduce hardware priorities into DRISC to prioritize selected real-time tasks. This is reflected in both the once-off procedure of concurrency creation and optimized dynamic thread scheduling. The results from the time-multiplexed execution of both the normal task and periodic real-time benchmarks highlight the benefits of the proposed strategy, and prove the wisdom of reserving thread slots instead of cores.