APPLE-CORE

Architecture, Paradigms and Programming Languages for Efficient programming of multiple cores

***KEYWORDS**: concurrency, programming model, compilers, microprocessors, multi-thread, many-core*

Background and Motivation

Unlike many other projects that adopt an incremental approach to processor design, the Apple-CORE project takes a holistic approach to multi-core architecture. The aim is to make multi-core mainstream and the project focuses on various key issues. First we tackle the problems of efficient concurrency management in microprocessor architecture, whilst at the same time maintaining what is one of the key characteristics of commodity processors, namely binary compatibility across generations of processors. In this we have been extremely successful, using dataflow principles in an imperative setting and using distributed shared memory with a relaxed consistency model.

To do this has required a bold solution: i.e. by defining a concurrent execution model of sufficient generality, fixing it and then embedding it into the core's ISA to provide hardware support for the operating system kernel. The benefit of the approach is efficient thread creation, synchronisation and context switching, supporting hundreds of threads per core. Data-driven scheduling provides asynchrony in instruction execution tolerating a significant amount of latency in non-local operations such as those involving external memory. With 100s of threads it is possible to tolerate latencies of up to a 1000 cycles, without impacting efficiency in the use of pipeline cycles. This makes it feasible to implement a cache-coherent, shared memory architecture on chip that has the potential for long latency access while searching a hierarchy of directories for a given cache line. The benefit of the cache protocol implemented on chip (COMA) is that data migrates transparently to the point of use.

This approach to many-core processors is disruptive requiring a significant initial development of compilers and operating system support. However, in adopting this approach, we ensure that there is future continuity following its adoption. Current approaches to continuity are provided by capturing sequential binary code and adapting it automatically by extracting concurrency during its execution. The Apple-CORE approach captures maximum concurrency in binary code and adapts that by applying a sequential schedule if necessary during execution. This means the same binary code will run on an arbitrary number of processors, from a virtual core of one thread up to a many multi-threaded cores.

A key principle in the Apple-CORE project is the separation of concerns between concurrency capture and the mapping and scheduling of that concurrency. Thus the high level compilers attempt to capture all of the concurrency exposed in programs and a combination of hardware and core compiler dynamically match this concurrency exposed to the parallelism of the available resources at execution time. The core compiler is based on GCC and compiles a new language that exposes the same concurrency model adopted in the core's ISA. This language is not intended as the end user language (except for system level programming) but instead acts as a target for other high-level parallelising compilers. Thus for applications programming we isolate the user from managing concurrency issues. We support deterministic programming languages such as sequential C (which is auto parallelised to maximum concurrency) and data parallel and functional languages, for example, Single-assignment C, which supports both these paradigms. The combination of run-time

system and features designed into the hardware auto-sequentialise the concurrency exposed from these compilers.

Objectives

The overall objective of Apple-CORE is to demonstrate that this separation of concerns between concurrency capture and scheduling in the SVP concurrency model can be implemented efficiently in a multi-core environment. Moreover, to demonstrate that it constitutes a viable target for compilers that automate the generation of concurrency. The goal is to make multi-core and parallel programming mainstream. This requires all concurrency problems to be managed automatically and dynamically in the implementation of the SVP run-time system, either in software, or in hardware.

A second major objective is to demonstrate that the implementation of this model in the ISA of RISC-like processor is not unnecessarily restrictive and to evaluate many-core chip designs that demonstrate the flexibility required in a general-purpose processor and its operating system. To this end it must provide a range of system services in a secure environment and be applicable to a wide range of applications. The approach promotes a space-sharing view of resources in the chip's OS rather than a time-sharing view.

To achieve this three execution platforms have been developed and are being used for evaluation:

- several software implementations of SVP of varying efficiencies, this initially provided functional verification of our SVP code but is currently being migrated towards an efficient run time system to port the SVP methodology onto a range of current multi-core and distributed systems;
- software emulation of a Microgrid of SVP cores with I/O to an external interfaces and a full set of system services able to execute SVP binary code and provide a cycle-accurate simulation of the execution time expected from the implementation of full-custom SVP multi-core chip;
- a soft-core prototype of an SVP core based on the LEON3 open-source core, which has been implemented in an FPGA and hence constrained by a different set of technological constraints.

Progress to May 2011 (end of Project)

The Apple-CORE project is divided into a number of technical work packages (WP2..6), each of which has the goal of performing both the enabling research and the tool development to support a particular aspect of the infrastructure for a many-core processor based on the Apple-CORE project (see Figure 1). This includes compilers, emulators and even an FPGA implementation of a single multi-threaded core.

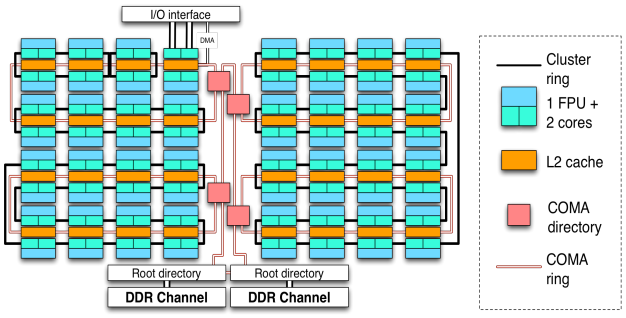


Figure 1. Example Microgrid configuration comprising 128 cores, configurable into clusters comprising a power of 2 processors. 1M Byte on-chip memory and 2 DDR3 2400 memory interfaces.

At the end of the project we have achieved all of our objectives, although there is still further evaluation and optimisation to be undertaken. However, to achieve this we have released the resulting tool chain to the public with open source licences, to enable further development and evaluation. More details of the tools and infrastructure are given by work package below.

Work package 2 is responsible for benchmarking and dissemination activities. It has developed an automated system for benchmarking (http://unibench.apple-core.info) where all of our evaluation activities can be accessed. The public can access this system with user/password=guest/guest and view all of the benchmarking activities for this multi-core processor. In addition to simple kernels we have also evaluated complete applications and Figure 2 shows the speedup obtained for the game of life application, which exploits data concurrency over chunks, list processing across chunks and requires exclusive updates of certain data structures.

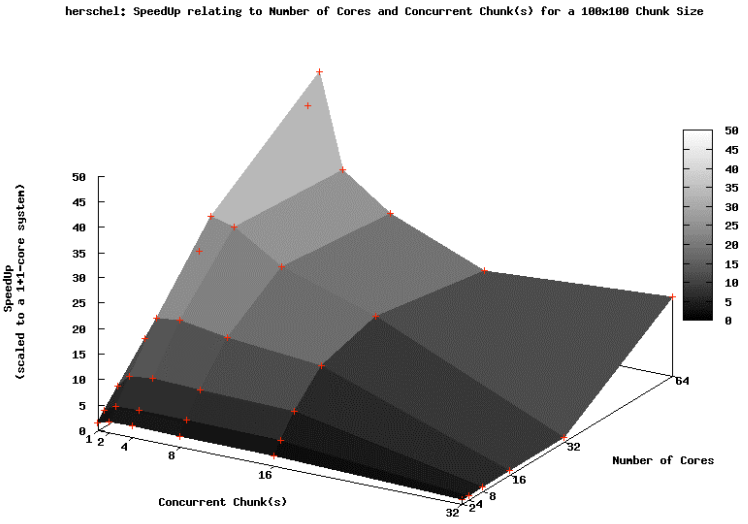


Figure 2. *Game of life speedup against single-core, single thread execution.*

Besides the academic publication WP2 has focussed on dissemination activities with a special emphasis on the difficult task of selling this disruptive approach to processor technology to potential industrial partners. Several companies and European Institutions have been contacted and an exploitation roadmap has been developed as a result of that activity.

Work package 3 is responsible for developing a parallelising compiler from the C language to target the Microgrid core compiler. The recently released 1st version of this compiler not only parallelises independent loops but also manages to obtain speedup from parallelising loops with multiple dependencies, even though the architecture only supports single dependencies between threads.

Work package 4 has investigated the potential for compiling high-level programming languages onto the Microgrid architecture. A back-end for the functional array programming language SaC has been generated. In the course of doing so various strategies for the selection of exploitable data-parallelism have been examined and experimentally evaluated. Several novel program transformation were devised that aim at minimising the inevitable overhead due to the semantic gap between language and architecture.

We have also implemented task-parallelism to exploit fine-grain concurrency on the Microgrid architecture. We identified the memory management as the main performance critical issue. A new memory interface has evolved as a result of a significant amount of hardware-software co-design. Based on that interface, a novel heap manager was invented. As a counterpart in the runtime system, we also developed a novel approach towards non-deferred garbage collection. It should be applicable for a wide range of languages and many-core architectures, not exclusively SaC and the Microgrid.

Furthermore, we have investigated the interplay between data-parallelism and functional parallelism. We found that effective heap management requires several different techniques for non-deferred garbage collection to co-exist. We developed and evaluated a novel multi-modal technique of reference counting that reflects this.

Work package 5 is responsible for the full system emulation of the many-core Microgrid. It has produced a timing-accurate emulation of a configurable multi-core chip with standard I/O and memory interfaces (Figure 1). It has also provided the core compiler and a set of system services.

The emulation simulates the execution time expected of a full-custom chip. This emulator supports a Posix-style interface with system services, such as I/O, the dynamic allocation of both memory and processing resources, and perhaps most importantly, solutions to resource deadlock on the exhaustion of these resources. To support the OS, principle solutions to security issues, i.e. protecting memory and cores/threads against unauthorised use has been proposed and reported again using a combination of hardware and software techniques.

Work package 6 has extended the LEON3 SPARCv8 instruction set by ten new machine-level instructions that support microthreaded assembly-level programs that are output of the µTC compiler. The new processor, called UTLEON3, has been successfully implemented and tested in Xilinx FPGAs. It is backwards-binary compatible with the LEON3 processor. In the microthreaded mode UTLEON3 tolerates latency of memory access instructions and long-latency arithmetic instructions, and on selected simple DSP kernels it outperforms LEON3 by 30% for small memory latencies (UTLEON3 performance is even better for higher memory latencies).

In addition UTLEON3 supports execution of families of threads in a special hardware accelerator. The execution is transparent to the assembler programmer, the actual execution place (CPU pipeline or HW accelerator) as well as all data transfers and family synchronization is executed in hardware. This implementation of the HW accelerator together with latency tolerance of microthreading shows one possibility how HW accelerators can be connected efficiently to a processor pipeline.

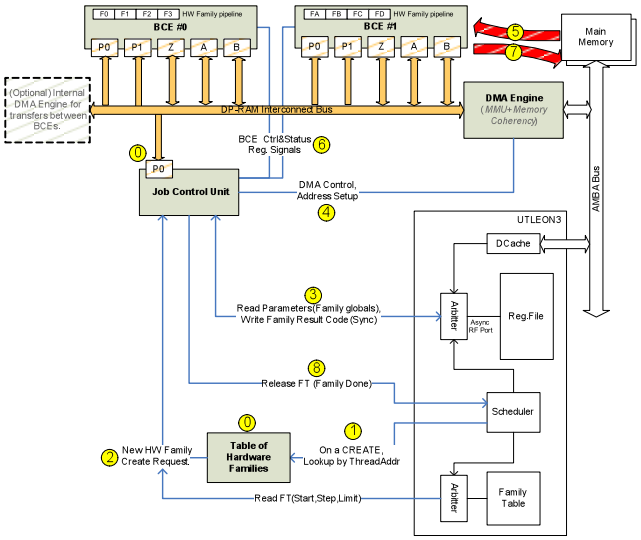


Figure 3: UTLEON3 - HW accelerator for families of threads.

Main Results

A number of significant results have been achieved by the Apple-CORE at completion.

- Perhaps the most significant result is the demonstration that it is indeed feasible to implement concurrency primitives in the core of a many-core chip without impacting its generality and to achieve the goal of binary compatibility across generations of processor products. This has been achieved by developing compilers that capture maximal concurrency and mechanisms to allow that concurrency to be dynamically adapted to the hardware resources available at execution.
- Using this approach we have demonstrated binary compatibility of code regardless of the number of concurrency resources available, extremes of latency tolerance that support all non-local operations, including access to off-chip memory and space sharing in our operating system kernel at a rate that exceeds the normal time-slice of a conventional operating system (i.e. at MHz rather than KHz rates),
- We have shown that it is possible to program this architecture from high-level languages, both from legacy sequential C to support a wide variety of legacy applications and also from SaC, a languages that captures data parallelism within a functional programming setting. Our initial evaluation of these compilers has shown that we can obtain good performance, despite the semantic gap and the strategy adopted in Apple-CORE of expressing maximal parallelism and having the system sequentialise when necessary, although this was not achieved without considerable effort.

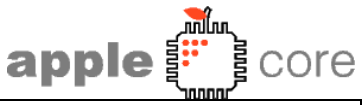
- Finally we have real hardware available in an FPGA implementation of a UTLEON3 soft-core that supports the execution of the microthreaded binary programs. The implementation first proved that the hardware support for microthreading can be implemented with a reasonable efficiency in current technologies. The processor supports zero-cycle context switching for switches induced in the fetch, decode or execute stages. This multi-threaded soft core also allows transparent execution of code in hardware using a programmable hardware accelerator.

Expected Impact

The expected impact of this project can be found in two areas, namely in concurrent processor design and in concurrent system software, where the former is significantly more difficult to exploit than the latter. Although embedding concurrency in the processor architecture has been demonstrated to be general and efficient for automatically managing fine grain concurrent software, it is unlikely that this approach will be adopted up by a mainstream commodity processor manufacturer in the short term, as the approach is not incremental. The exploitation strategy has to be long term and for this reason, the tools have been made available in the public domain and engaged several manufacturers in targeting the tool chain developed to conventional multi-cores processors (see below). There is however, an opportunity for impact from the processor design in niche markets and we have already explored the exploitation of this technology in space through a proposed collaboration with the European Space Agency. The interest for space applications is two areas: the exploitation of the concurrency model for real-time applications, where the hardware supported threads provide very low overhead context switching; and in fault-tolerance and reliability, where the concurrency model would support flexible and transparent implementation of redundant execution (either in space or in time).

The second potential impact comes from the SVP tool-chain, as the tools can be retargeted to cores that do not necessarily support this concurrency API in the core's ISA. It is possible to implement the same techniques at a coarser level of granularity in any multi-core processor architecture, provided that the interface supports the same strategy of reducing concurrency levels, so that concurrency overheads can be amortised. Currently the project is engaging with Intel in implementing SVP one its SCC system (a 48-core chip) and also with Oracle who already have a fine-grain multi-threaded multi-core processor architecture.

The research on SaC benefits compilation to other architectures not only the Microgrid. Insights gained apply to all runtime systems of high-level languages with implicit memory management that operate in a highly multithreaded environment. There has been increased industrial and academic interest in using the SaC technology for performance intensive applications. Further work on making this technology available for legacy codes via compilation from languages with existing industrial applications is underway, with a consortium for a STREP project with 50% industrial involvement. The creation of a spin-off company for maintaining the tool-chain is also in progress.



Project Coordinator *Professor Chris Jesshope* **Project Technical Manager** *Professor Chris Jesshope* *University of Amsterdam* *C.R.Jesshope@uva.nl* *http://www.apple-core.info*	**Partners**: *University of Amsterdam (NL),* *University of Hertfordshire (UK),* *Institute of Information Theory and Automation (Czech Republic),* *University of Ionnina (Greece),* *Associated Compiler Experts (NL),* *Aeroflex Gaisler (Sweden).*	**Duration:** *36 months* **Start:** 2007.11.01 **Total Cost:** *€ 2.750.000* **EC Contribution:** *€ 2.100.000* ***Contract Number:*** *INFSO-ICT- 215216*