Evolution of Parallel Architecture Targets

Moderator: Henry Dietz[0000-0002-5878-881X]

Panelists: Hironori Kasahara, Movahhed Sadeghi, Ishan Thakkar

University of Kentucky, Lexington KY 40506, USA
hankd@engr.uky.edu

Abstract. In the early days of the LCPC workshops, a very large fraction of the work being done targeted either vector machines or massively-parallel SIMD (single instruction stream, multiple data stream). This was partly due to their dominance in the high-performance computing market, but those targets also were an excellent match for the compiler analysis of the day, which centered on nested `DO` loops in Fortran programs. At various points in the history of the workshop, focus has shifted to a wide range of other types of parallel architectures. With these shifts came some new languages and many different compiler analysis methods; for example, some emphasized inter-procedural analysis use while others became focused on data layout and still others dealt with timing properties and scheduling. An increasing variety of efficiency-critical features (ECFs) are found in modern computers, and the concept of "dark silicon" and cost-effectiveness of attached parallel processors has been producing hybrid computing systems that incorporate multiple architectures. In the highly-connected world we now live in, it also may be appropriate to consider architectural targets that are literally "outside the box" – targeting systems that are partially local, but explicitly include remotely-accessed computing facilities. This panel examines how parallel computer architectures are evolving with the goal of suggesting appropriate future research directions for the LCPC community.

Keywords: Computer Architecture, Vector, SIMD, MIMD, GPU, FPGA, Quantum Computing, Optimizing Compilers, and Parallelizing Compilers.

1 Introduction

This panel is bringing together … to discuss the evolution of computer architecture. The discussion is focused not on the architectures themselves, but on what the evolution of architectures means in terms of guiding directions for future research within the LCPC community. Thus, the goal is to identify how language and compiler research should be moving to best support coming generations of computer systems.

The panel discussion at LCPC (and this paper) begins with a brief introduction by the moderator followed by ten-minute position presentations from each of the panelists. Each panelist independently determines the content of their position presentation and submits a section for this paper after the workshop. To ensure collection of opinions on some specific topics, the moderator prepared and distributed this "Intro-

duction" section of the paper before the workshop, giving three prompts to be addressed by each panelist. The conclusion is written by the moderator after the workshop, summarizing the discussion that followed the position presentations.

1.1 Efficiency-Critical Features

The world has grown to depend upon continuous, mostly exponential, performance improvement over time for computer systems – but that would not happen if computer architecture did not evolve. An efficiency-critical feature (ECF)[1] is any architectural attribute of a computer system that must be used effectively in order to obtain good performance. Major increments in system performance usually are enabled by introduction of new ECFs. As computer architectures have evolved, ECFs have included:

- *Bit-slicing and multiple-word-precision arithmetic*
- *Floating-point arithmetic*
- *General register files*
- *Support for subroutines, functions, and recursion*
- Flat and structured memory address spaces and pointers
- Pipelined parallelism
- Vector parallelism and array processors
- SIMD (single instruction stream, multiple data stream) parallelism
- MIMD (multiple instruction stream, multiple data stream) parallelism
- Multithreading (originally known as barrel processing)
- Caches and memory hierarchy, coherence and consistency models
- Shared and distributed memory
- Atomicity, directed and barrier synchronization, and transactional memory
- VLIW (very long instruction word) and SuperScalar parallelism
- SWAR (SIMD within a register) parallelism
- Message passing and aggregate functions/collective communications
- DSPs (digital signal processors)
- GPUs (graphics processing units)
- FPGAs (field programmable gate arrays) and reconfigurable systems
- Dark silicon and advanced power management
- Quantum computing (entangled superposition is parallel processing)

The first few of the above ECFs (listed in *italic*) don't necessarily involve parallel processing, but all the others generally do. Parallel execution is behind most of the performance increase that we have seen from computing systems over the last four decades. No doubt, the future will see computers evolve to incorporate many more ECFs, most involving parallel processing, that will require some level of language and/or compiler support.

Thus, the first prompt given to the panelists is:

What current or emerging parallel computer ECFs most desperately need new types of programming language and/or compiler analysis and transformation support?

1.2 E Pluribus Unum: Out of Many, One

Although lots of different types of ECFs will be seen as architectures continue to evolve, perhaps the strongest current trend is that systems do not have just one type of parallelism, but incorporate multiple somewhat distinct subsystems or attached processors each executing with different types of parallelism.

Given that different types of parallelism appear in different applications or portions of applications, it makes sense that the best speedup from parallel execution will result from having hardware that efficiently supports a variety of types of parallelism. Beyond that, there is the fact that some types of parallel architecture literally cannot stand alone. For example, much of the efficiency of a GPU comes from omission of hardware structures that would enable it to function as a fully general-purpose computer. Even more dramatically, quantum computation achieves exponential amounts of parallelism in operating on entangled superpositions (and uses quantum phenomena rather than a multitude of hardware modules to implement that parallel execution), but an entangled superposition is a very fragile thing, and values cannot be held for arbitrarily long periods of time – in sum, quantum computers currently implement only relatively shallow combinatorial logic. Thus, every quantum computer is a very special-purpose massively-parallel computing engine hosted and controlled by a conventional computer. Even Shor's famous "quantum algorithm" for finding the prime factors of a number[2] is primarily executed on a conventional machine that repeatedly invokes a quantum order-finding subroutine to narrow the search space.

Nearly all modern computing systems, from embedded microcontrollers to supercomputers, now have architectures combining various types of parallel hardware structures. At this writing, even microcontrollers costing just a few dollars contain pipelined, superscalar, SWAR-supporting, multi-core processors, and often augment that with a management processor on the same chip (typically an ultra-low-power processor that handles tasks like waking the main system in response to sensor inputs). In larger systems, combining similar features with a GPU is probably most common, and supercomputers tend to make distributed-memory MIMD systems with that mix within each node.

The second prompt given to the panelists asks them to gaze deeply into their crystal balls and answer:

What types of heterogeneous parallel architectures do you see becoming the most important combinations for compilers to target within future computing systems? Perhaps your answer differs for each class of computers: embedded microcontrollers, smartphones and tablets, laptop and desktop computers, servers, and supercomputers.

1.3 Everything Everywhere All at Once

As much as computer architecture has been evolving, a second effect has arguably been even more dramatic: everything is becoming continuously connected.

Traditionally, programming languages and compilers tend to target ***a machine*** – not a huge collection of loosely-connected systems. However, the concepts of Grid and Cloud computing, IoT (internet of things), and especially Edge computing are making the boundaries between computing systems increasingly unclear. For example, a $7 ESP32 IoT boardlet not only provides a surprisingly capable dual-core computer, but also provides both 802.11 Ethernet and Bluetooth wireless connectivity. The smartphones that so many of us carry everywhere are even better connected, and so are the computers on our desks and in our machine rooms.

Certainly, it is now common that programs are written in such a way that they trivially can be moved from a local machine to a remote system. This is what the concept known as Cloud Bursting is all about: moving an application to a cloud facility when local resources are insufficient either due to machine configuration or high load. Automatically having a compiler determine which machine, or set of machines, in a heterogeneous collection of computers should run a program by taking both architecture and current loading into account is an idea that the moderator worked on and crudely prototyped as early as 1993[3] – but automation of this type of analysis and runtime selection of targets has never become a common approach. Perhaps there are good reasons both for and against taking such an approach...

This notion of everything connected motivates the third prompt:

> ***Should the LCPC community, as language designers and compiler writers, no longer be thinking only about targeting a machine, but also about targeting the heterogeneous system that is only partly the machine we have near us? If so, state what the main research challenges are; if not, explain why not.***

2 Panelist: ...

A position statement contributed by each panelist...

5 Conclusion

To be written by the moderator after the panel.

References

1 Henry Dietz, “The Refined-Language Approach to Compiling for Parallel Supercomputers,” Doctoral Dissertation, Polytechnic University, Brooklyn, New York (1987), ISBN 979-8-3684-1591-8
2 Ekert, Artur, and Richard Jozsa. "Quantum computation and Shor's factoring algorithm." Reviews of Modern Physics 68, no. 3 (1996): 733.
3 H. G. Dietz, W. E. Cohen and B. K. Grant, "Would You Run it Here or There? AHS: Automatic Heterogeneous Supercomputing," 1993 International Conference on Parallel Processing - ICPP'93, Syracuse, NY, USA, 1993, pp. 217-221, doi: 10.1109/ICPP.1993.187.