Parallel Computing 75 (2018) 28–40

Contents lists available at ScienceDirect

Parallel Computing

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

Introducing the explicitly many-processor approach

János Végh

University of Miskolc, Hungary

ARTICLE INFO

Article history:
Received 3 July 2015
Revised 1 February 2016
Accepted 1 March 2018
Available online 9 March 2018

Keywords:
Many-core
Single thread
Performance
Neumann abstractions
Hybrid architecture
Explicitly many-processor approach
Computing principle

ABSTRACT

The deeper reasons of the present stalling in computing is scrutinized, and to enhance the single-processor performance, a new approach explicitly considering the presence of several computing units is introduced, as opposed to the presently exclusively used, 70-years old single-processor approach. The appearance of many-core processors, having many processing units in close vicinity to each other, requires to re-think some principles of computing. The goal of the approach is to enhance the single-processor performance using cooperating cores, rather than to introduce a new method for parallelization. Technically, it introduces a new control layer above the cores, a new intermediate execution unit called quasi-thread, a modified compiling method and object code for transferring parallelization information from the development system to the processor, and an on-demand self-organizing processor architecture. The resulting processors have more effective and more "green" architecture, considerably increased single-thread performance, allow for more deterministic real-time behaviour, new scheduling principles for multitasking, less operating system overhead, etc. Surprisingly, the resulting computing stack is upward compatible with the presently existing one.

1. Introduction

In the past years, the development of computing seems to stall. The single-processor performance, although is still slowly growing, in some cases is even decreasing. More and more design efforts are needed to cope with the growing complexity, and similarly, more and more Si gates must be used to implement it, greatly contributing to the dissipation problem, and leading to the age of "dark silicon" [1]. "*...computers have thus far achieved this goal at the expense of tremendous hardware complexity – a complexity that has grown so large as to challenge the industry's ability to deliver ever-higher performance*" [2]. For illustrational purposes, two representative manufacturers and randomly selected, publicly available data were used to derive the data[1] shown in the diagrams in Fig. 1.

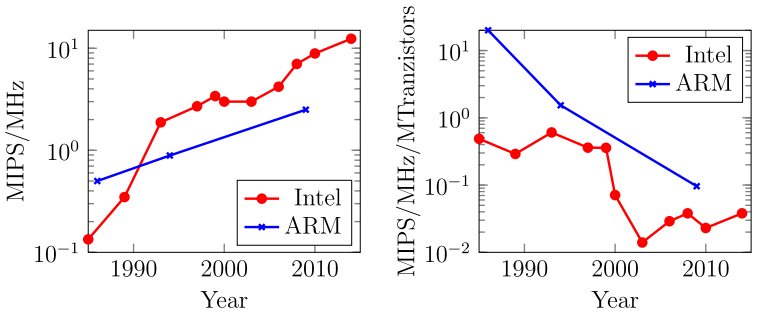
To quantitize the performance, one can find somewhat arbitrary merits [3]. To support our arguments, the MIPS value is used here. One can (mostly) separate the purely architectural improvements from the technological enhancement through dividing the MIPS value with the operating clock frequency (a kind of frequency-independent performance). This value is shown in Fig. 1a.

Another important question is, how efficiently the architectural enhancement can be implemented. This question can be (approximately) answered using a merit, which is given as the frequency-independent performance divided by the number

E-mail addresses: J.Vegh@uni-miskolc.hu, janos.vegh@unideb.hu

[1] Obviously, the absolute numbers must not be compared for the different architectures, but the tendencies are the same even for those different architectures.

https://doi.org/10.1016/j.parco.2018.03.001
0167-8191/



(a) Timeline of architectural enhancement of processing performance

(b) Timeline of efficiency to implement architectural enhancement of processing performance

Fig. 1. Timeline of architectural performance and implementation efficiency of computer processors.

of transistors used in the implementation. Although the number of the transistors slightly increases also because of the extended functionality, the main reason of the increase is adding more helper units to the processor to increase performance. This latter merit describes, how economically the needed performance enhancement can be implemented through architectural changes, see in Fig. 1b. The reciprocal of that value can be considered as a measure of architectural complexity. These diagrams suggest that the frequency-independent performance keeps rising, but more and more efforts (more and more complex architectures) are needed to keep the tendency. The hardware designers attempt to take over as much duties as they can (dozens of cores, instruction level parallelism, out of order and speculative evaluation, etc.), but there is no commonly accepted idea for making efficient really multi-processor hardware architectures: "*multicore and manycore development tool vendors and runtime systems cannot possibly support the virtually unlimited number of processor configurations*" [4].

The many-core/supercomputing direction of development offers not a real alternative for most cases: on this branch also a lot of programmers' efforts must be invested to organize and concert the work of the many processors, and also the parallel execution of the tasks must be designed in advance. However, "*parallel programs ...are notoriously difficult to write, test, analyze, debug, and verify, much more so than the sequential versions*" [5]. In addition, they also show up serious efficiency problems. The typical real-life programs show complex parallelization behaviour [6] and also the apparently massively parallel algorithms can behave [7] extremely ineffectively.

Since the beginning of computing, the *single processor approach* dominates in computing. In many-processor systems we do have many *central* processing units, and a peripheral must *interrupt* the execution flow of the processor, although some other core or processor would be able to do the job. Even today the many-processor computing bounds are referring to *Amdahl's law* although the title of the paper was "Validity of the *Single Processor Approach* to Achieving Large-Scale Computing Capabilities". When Hill and Marty [8] speaks about "multicore era", the performance is described by (a modified version of) Amdahl's law, although in the same study the "dynamic multicore" approach results in much better performance. Most researchers are convinced that "*Processor and network architectures are making rapid progress with more and more cores being integrated into single processors and more and more machines getting connected with increasing bandwidth. Processors become heterogeneous and reconfigurable. ... No current programming model is able to cope with this development, though, as they essentially still follow the classical van Neumann model*" [9].

Essentially, this is why most experts worry, requirering "*Reinventing computing*" [10], asking "*Computing Performance: Game Over or Next Level?*" [11], or thinking in "*Rebooting computing*" [12]. On the other hand, the new computing must be compatible to some extent with the old one. In order to *preserve compatibility with the existing computing*, the best way would be to modify the existing one in a way which allows taking into account the present-day achievements of technology as well as the expectations against computing.

The present paper will show how many-core processors, using a new approach to computing, can be used to enhance the performance of single-thread processes. Our goal is just to increase the performance of single processors utilizing a new principle of operation, focussing on the "cooperative solution" mentioned by Amdahl [13] and targeting the "two fundamental issues": latency and synchronization, identified by Arvind and Iannucci [14]. As a consequence of using the many computing resources inside the processor in a more efficient way, also increases performance of the parallelized computers.

In Section 2, the related work of potential anchestors are considered, both hardware and software activities. The Explicitly Many-Processor Approach is introduced in Section 3. This section describes the details of the new operating principles and their features, as well as compares the method to its predecessors, emphasizing similarities and dissimilarities. The abstract ideas are demonstrated on a simple C language code example in Section 4, where most of the parallelism can be discovered using the methods used in the related works, and the result can be refined using the methods suggested in this work. A deeper perception of the operation of such a system can be acquired in Section 5, where the "object code" of an example