Available online at www.sciencedirect.com
ScienceDirect
Physica D 237 (2008) 1282–1301
PHYSICA D
www.elsevier.com/locate/physd

The foundation of self-developing blob machines for spatial computing

Frédéric Gruau[a,b,c,d], Christine Eisenbeis[b], Luidnel Maignan[b,*]

[a] LRI - Université de Paris-Sud 11, bâtiment 490, 91405 Orsay Cedex, France
[b] Inria Futurs Saclay - Parc Orsay Université, 4, rue Jacques Monod, 91893 Orsay Cedex, France
[c] Laboratoire d'informatique, de robotique et de microélectronique de Montpellier, 31 rue Ada, 34000 Montpellier, France
[d] University of the West of England, Frenchay Campus, Coldharbour Lane Bristol BS16 1QY, United Kingdom

Available online 4 April 2008

Abstract

The current trend in electronics is to integrate more and more transistors on a chip and produce massive hardware resources. As a consequence, traditional computing models, which mainly compute in the temporal domain, do not work well anymore since it becomes increasingly difficult to orchestrate these massive-scale hardware resources in a centralized way. Spatial computing is a unifying term that embodies many unconventional computing models and means computing on a relatively homogeneous physical medium made of hardware components, where the communication time is dependent on the Euclidean distance between the components (locality constraint). This constraint makes the programming for high performance significantly more complex compared to classical non-spatial hardware because performance now depends on where computation happens in space (mapping problem). Blob computing is a new approach that addresses this parallel computing challenge in a radically new and unconventional way: it decouples the mapping of computations onto the hardware from the software programming while still elegantly exploiting the space of the underlying hardware. Hardware mapping of computations is done by a physical force-based approach that simulates forces between threads of computation (automata). Attractive forces are used to keep automata that need to communicate with each other closer while repulsive forces are used for load balancing. The advantage of these primitives is that they are simple enough to be implemented on an arbitrary computing medium. They form the basis of a runtime system (RTS) that transforms an arbitrary computing medium into an easier-to-program virtual machine called the blob machine. The basic objects of the blob machine are those automata, and the instructions let automata create new automata in specific ways so as to maintain a hierarchical organization (which facilitates both the mapping and the programming). We detail the basic instructions of the blob machine and demonstrate their confluence. Programming a spatial medium to perform a given algorithm then boils down to programming the blob machine, provided the RTS is implemented on it. The advantage of this approach is the hardware independency, meaning that the same program can be used on different media. By means of several examples programmed using a high-level language description, we further show that we can efficiently implement most current parallel computing models, such as Single Instruction Multiple Data (SIMD), data parallelism, "divide-and-conquer" parallelism and pipelining which demonstrates parallel expressiveness. On sorting and matrix multiplication algorithms, we also show that our approach scales up optimally with the number of basic hardware components.

Keywords: Spatial computing; Distributed algorithm; Self-developing network; Computer architecture; Computer language

1. Introduction

1.1. Motivation: Scalability and expressiveness

An important part of computer science is devoted to designing a context that is conducive to programming the computer and obtaining results efficiently. This covers many areas of research, organized in layers, including hardware, architecture, machine language, and high-level language. The lowest layer deals with the physical properties of physical entities to provide a preliminary set of manageable components and rules, and the highest level attempts to provide an abstract representation of the former so that the programmer can describe the desired task in an expressive way, while offering the best feasible performance.

The scalability problem. At the lowest physical level, technology already produces chips with billions of transistors.

* Corresponding author.
E-mail address: luidnel.maignan@gmail.com (L. Maignan).

0167-2789/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.physd.2008.03.046

As a result, the number of processing elements (PEs) is steadily increasing and research in unconventional computing currently focuses on building hardware on a scale that outreaches today's technology. Given such magnitudes, scalability becomes imperative at the high level: as the number of PEs increases, does performance increase accordingly? This paper advocates that the conventional ways of designing parallel hardware architecture are not appropriate for scaling to arbitrary large size because too many of the physical level properties are lost in abstraction. For example, many classical features of parallel architecture, such as shared memories or all-to-all routers, design out the notion of space to establish a single Uniform Memory Architecture (UMA). In shared memories the actual location of data is not an issue. With all-to-all routers, any two PEs are considered as being close together. But the performance of these features cannot be scaled, which is necessary in larger systems where communication time (to access a single memory or communicate with another PE) increases with size. The time required for a signal to travel the length of the wire is not taken into account.[1] The spatial computing framework identifies space as the key physical property and proposes to organize hardware resources as a spatially extended homogeneous computing medium in order to take space into account and achieve greater scalability. Computation and data must be distributed in 2D or 3D space and the particular spatial arrangement must be closely articulated with performance.

The expressiveness problem. At the highest level, the computing medium must be programmed, i.e. the behavior of each individual processing element (PE) in space must be described. This is a difficult task in comparison to programming sequential machines: there is no centralized control, no overall coherent memory image of the machine configuration, and no global clocking. Because of locality in space, each PE communicates only with its nearby neighbors. Most programs running on spatial computers implement a single purely spatial algorithm where input and output data are located in space and data computation can also take place naturally in space. This is clearly insufficiently expressive when aiming to use spatial computing to solve complex tasks involving different algorithms and data structures. The task is even more difficult when performance is important.

The blob machine concept proposes to solve the programming problem by using a vertical approach to spatial computing, i.e. by proposing two levels that gradually abstract space while never completely ignoring it. Programming is carried out on an intermediate virtual machine called the blob machine, whose primitives are based on physics simulation and are sufficiently simple to be implemented on an arbitrary computing medium. On the one hand this allows the user to program traditional parallel algorithms without worrying about the exact spatial location of data and computation. On the other, if the programmed task graph is simple enough, as is the case for a planar graph, then the runtime system can efficiently map it in 2D space.

The rest of this article is organized in three introductory subsections followed by two sections. Section 1.2 outlines state-of-the-art spatial computing as a hierarchy of horizontal layers from hardware to software. Section 1.3 informally introduces the blob approach and its main features of interest. The self-mapping property, a significant facet of the blob model, is described in Section 1.4. After that, Section 2 gives a formal definition of a simplified blob machine, i.e, the binary blob machine and the state of the art of its implementation. The model semantics are described in detail and semantic confluence is demonstrated. An explanation follows describing implementation of the blob machine on a computing medium and defining a complexity model, i.e., "dDcomplexity", used to measure performance in the examples of blob execution presented in Section 3. Those examples have been chosen to cover a wide range of parallel algorithms. The purpose is to establish the feasibility of programming and assess the resulting efficiency. Optimal time and space complexity can be achieved, as long as implementation meets dDcomplexity requirements.

1.2. Background on spatial computing

Spatial computing is an umbrella term that groups together different approaches, all based on the observation that future computing platforms – whether very-large-scale integration (VLSI), bio, or nano – will consist of a vast number of Processing Elements (PEs) homogeneously embedded in 2D or 3D space, where the magnitude involved obliges the programmer to incorporate the *locality constraint*, where each PE has a specific location in space, and communication time is a function of Euclidean distance in that space. For example, in the (classic) VLSI complexity model [1], this relationship is linear. Communication costs have always been a major issue in parallel computing. But scaling up to an arbitrary large space is rarely considered as an option, where communication must be optimized by taking into account physical distance. For example, a black-box router can be implemented efficiently in all-to-all communication, but it abstracts away spatial location. Architectures embedded in space, referred to here as *"computing media"*, include not only *regular* classic models, such as cellular automata, systolic arrays and FPGAs,[2] but also *irregular* models where the constraints of lattice tiling of space and synchronism in time are relaxed, as exemplified in the amorphous computing model [2]. Spatial computing was the subject of a recent workshop [3]. A complexity model of computing media called "spatial machines" is presented in [4].

Spatial computing calls for a departure from computing in time, which uses a conventional centralized programming approach with a step-by-step modification of a given overall state. Intuitively, to exploit space, computation must be

[1] Consider the example of a router where spatial location has been abstracted away and the router diameter is the measure of router performance. If the communication time between any pair of PEs does not depend on the communicating PEs, it necessarily depends on this diameter, which represents the worst case.

[2] Field-Programmable Gate Array. An FPGA architecture can be reconfigured on the fly and therefore be adapted to the dynamic features of the program it executes.