Many-core and heterogeneous architectures:

programming models and compilation toolchains

PhD Candidate: Francesco Barchi

Supervisors
Prof. E. Macii, Supervisor
Prof. A. Acquaviva. Co-supervisor

Summary

In the last ten years, we have witnessed a revolution in technology. The achievement of the physical limits of silicon lithography has required several architectural innovations, both in the development and integration of dedicated accelerators and in biomimetic systems that seek to change the traditional computational model radically.

Nowadays, heterogeneous architectures in a single silicon slice integrate radically different computational models such as MIMD and SIMD, but also programmable logic (FPGA).
In the same period Machine learning has developed enormously, and with the advent of Deep Learning the demand for dedicated hardware has led to an explosion of Cambrian accelerators and specialised architectures. In the field of biomimetic systems, the scientific community and companies, like IBM and Intel, have shown interest in neuromorphic systems. A neuromorphic system can emulate at hardware or software level the electrical behaviour of neural networks.

In this context of heterogeneity and new paradigms in computing devices, my research has focused on the programming models of these architectures and the optimisation of their resources. My work has focused on neuromorphic architectures based on manycore architectures and heterogeneous embedded architectures. Neuromorphic architectures offer a new computational paradigm that requires the careful use of communication resources and the development of the entire software stack to provide an appropriate programming model capable of masking the internal architecture complexity to the user. Heterogeneous architectures also require masking their complexity to the user. In this case, I worked on the strengthening of the current compilation chains through the use of deep learning to perform code analysis and to extract information useful to make complex decisions now delegated to the programmer, for example choosing the most suitable calculation unit for code execution.

In the first case, I worked on SpiNNaker within the European Human Brain Project (HBP). SpiNNaker is a manycore neuromorphic architecture for which I developed a network resource optimisation system and a new programming model based on message passing. Specifically, I developed a toolchain for the mapping of Spiking Neural Network (SNN) within SpiNNaker measuring its performance based on the reduction of communication on the architecture and developed a whole software stack to implement MPI. In the second case, I worked on source code classification via deep neural networks. In particular, I built a source code classifier in the intermediate representation of LLVM able to discriminate the most suitable calculation unit for quick execution of the analyzed code.

Programming tools and middleware for manycore neuromorphic platforms

My achievements obtained in research on programming tools and middleware for manycore neuromorphic platforms are divided in two categories: optimization of communication resources during SNN simulations and development of a software stack for the implementation of a parallel programming model based on message exchange. I proposed a methodology for profiling densely interconnected neuromorphic multi-chip many-core platforms for real-time SNN simulations. The methodology has been used to characterise reliability issues in the SpiNNaker platform, impossible to investigate using a biological network. I designed custom SNN configurations to unveil both local and external network traffic issues. I prove that one of the causes of unreliability was due to packet conflicts in the internal router tree related to traffic congestion. This unreliability can be due to simultaneous usage of communication links of a router. Results show that, with a good neuron population placement, it is possible to improve simulation reliability by decreasing the total number of packets exchanged. We have modelled the mapping problem of complex directed graph (SNN) into the SpiNNaker processors-mesh. We have identified and tested 4 methodologies to solve the problem. The *Naïve* method (a simple heuristics), the *Spectral* method (uses the graph eigendecomposition to obtain a planar representation of the SNN graph and performs the node association with the chip mesh through an ILP formulation), the *Scotch* method (uses the *Dual Recursive Bipartitioning* heuristic), the *Simulated Annealing* method (well-known procedure to minimise a cost function). We have defined the cost function of the placement problem using the synaptic elongation. We have chosen the cortical microcircuit as our benchmark network, and after performing several tests we highlight the performance of each method. The Spectral method was implemented in GHOST, a Python module compliant with the sPyNNaker toolchain in order to demonstrate the effectiveness of the developed mapping approach with respect to random neuron placement. Finally, comparisons were made between configurations produced by PACMAN and GHOST. From these simulations was evident that GHOST is capable to reduce the number of used cores, results in lower R2R traffic, 96X when GHOST is adopted.

The architecture provides an inefficient unicast communication protocol, unsuitable for the development of a communication library such as MPI. For these reasons, we developed a communication middleware (MCM) based on the Multicast protocol. We reduce the complexity of the internal transmissions, by implementing unicast communications avoiding the supervision of the monitor processor. On top of MCM we designed the Application Command Framework (ACF), and Application Command Protocol (ACP) a new method to be adopted at the application level for spreading commands and manage the private memory of the chip processors. It provides a abstraction level of the memory (Memory entities). Users can easily access to all application Memory entities. To prove the advantages of our ACF we modify two SpiNNaker application enabling them to use our library. The first application is used during the configuration phase of SpiNNaker board, and the second application is a neuron model used during the SNN simulation phase. In the first application, we demonstrated the advantages introduced by ACF in the run-time feeding of configuration applications. The use of ACF can speed-up host-to-boards data transmission during the configuration of SpiNNaker platforms. Exploiting the concurrency of the system we have been able to get an improvement of 3 times on the data forwarding inside the board. In the second application, we demonstrated the run-time flexibility introduced by the ACF embedded in the neuron model application, implementing two different real simulation scenarios:
i) a two-phases SNN-Classifier designed for discriminating the handwritten number and
ii) a chain of neurons with run-time re-configuration parameters. Both the implemented applications were demonstrated to be flexible, scalable and expandable. Lastly, I described an implementation of the MPI paradigm on the SpiNNaker neuromorphic platform exposing a

programming model for the development of parallel applications without knowledge of the interconnections between the computing units of the underlying architecture. In the case of SpiNNaker, the implementation of MPI take advantage of the technology offered by on-chip routers, obtaining efficient communication by using the ACF and memory entities. This software stack creates a simple working framework offering a universally known programming model capable of making the SpiNNaker architecture available for a wide range of applications. We benchmarking our MPI implementation, showing its linear scaling performances executing two MPI programs. The first application was an N-Body simulation where 2k particles were simulated on 240 processors with a speed-up of 194x and an efficiency of 80% when compared to the serial version running on a single SpiNNaker core. We also presented an MPI implementation of a DNA sequence matching algorithm. Results show that the scalability of the SpiNNaker board reaches an ideal profile, 98% of efficiency, when using more than 100 processors, a 90% efficiency using 600 processors, reaching 88% efficiency when all 767 application processors are used.

Programming tools for heterogeneous platforms

My achievements obtained in research on programming tools for heterogeneous platforms was the implementation of a source code classifier, analysing a LLVM-IR code with deep neural networks. The objective of this research is to provide the current compilation chains of a more complex code analysis mode capable of making complex decisions that would otherwise be difficult to codify in a set of rules. We have therefore trained deep learning models capable of automatically learning features from source code. In particular we wanted to show that it is possible to analyze code in intermediate representation. I developed a LLVM-IR code classifier using two neural network models (CNN and LSTM) and make some comparisons between them. The code sequences before to be passed to the neural network classifier need to be transformed for condensing language elements in a restricted set of keywords (tokens), filtered for removing less informative tokens, and then transformed in numbers. We evaluated the performances of our LLVM-based classifier using a dataset of OpenCL kernels labeled with the best compute units in terms of runtime (CPU o GPU). Then, we explored the hyperparameters of a CNN model in order to obtain a reference model.
We explored 368 different hyperparameters configurations, reporting a statistical analysis of the results obtained. We compared the best configuration of hyperparameters for the CNN with the RNN-based network for different source code preprocessing and token filtering strategies, evaluating classification accuracy, MCC and Speed-up. Results confirm that features extraction from LLVM-IR is a valuable strategy for analysing sources without dealing with complex high-level constructs, and it can be done keeping all the information required for performing classification tasks in the context kernel-device mapping.

Overall this work has allowed exploring the potential of these new generation architectures and will be useful technologies for future developments.