OpenSPL: Revealing the Power of Spatial Computing

The OpenSPL Consortium, Dec 2013.

***Abstract*—Computing in space is emerging as an attractive complement to temporal computing, i.e. multicore processors, for building future scalable computing systems. Efficient implementations of complex structures that could previously only have been achieved with custom hardware development can now be accomplished using compact yet powerful software methods. As a consequence, we are experiencing the convergence of hardware and software. Scalability, efficiency and complexity limitations of temporal computing with multiprocessors are driving a need for radical innovation in the design of future computer systems.**

The Open Spatial Programming Language (OpenSPL) has been developed to enable domain experts to generate optimal computational structures for application specific challenges bridging the previously separate hardware and software domains. With OpenSPL, developers can now create systems following traditional software development lifecycle (SDLC) principles for both ultra high throughput computing and ultra low-latency communication while also reducing the carbon footprint by means of minimizing energy consumption, cooling and overall datacenter space.

***Keywords*-spatial computing; high performance computing; low-latency; OpenSPL**

I. Introduction

Spatial Computing exploits the advantages enabled by the steadily growing numbers of computational resources on a chip. From 2003 to 2013, the number of transistors in modern microprocessors went up from 400M (Itanium 2) to 5 Bln (Xeon Phi). However, over the same period memory access latency only improved by less than 3 times.

Since there is now so much more space, while time can not be compressed any further (processor clock has been stagnating at few GHz), the insatiable demands of industrial growth suggest that it is just a matter of time until all mission critical computations embrace the Spatial Computing paradigm in order to supply the expected annual performance improvements for computer systems.

Spatial computing systems support and deliver the highest degree of parallelism without the need to parallelize the algorithm. In spatial computing all units are depending on each other and at the same time all operations happen in parallel. In essence, all of the operations can be customized and exist in space, while the data to be processed just flows through the different stages of an ultra-large custom datapath. All computation starts as a maximally unrolled dataflow graph which is then folded into the ever growing space on flexible spatial computing substrates. The spatial computing approach brings a clear advantage over classical temporal computing systems that execute a temporal sequence of operations on a limited number of units (cores).

Spatial computing systems decouple the program's control flow and data flow, similarly to decoupled computer architectures [8]. Furthermore, by limiting ourselves to static predefined computations on large datasets, the penalties of dynamic memory hierarchies and dynamic branch management can be completely removed. OpenSPL facilitates efficient computation by generating static spatial kernels, each of which can have a single clock. The dataflow and controlflow components of spatial computers are fully programmable allowing adaptations of the machine architecture and number representation to best fit the algorithm structure and the specific numerical demands of the application.

The OpenSPL machine model consist of a Spatial Computing Substrate (SCS), i.e. a specific hardware technology, and may contain flexible arithmetic units, customizable interconnect and a set of memories. In the standard, we differentiate three independent memory types: Scalars, Fast (small) Memories (FMEM) and Large (slow) Memories (LMEM) with different sizes, bandwidths and latencies. SCS supports the implementation of one or more static computational *kernels* that are interconnected via flows of data. The OpenSPL execution model assumes the SCS executes as a single entity while all its internal kernels and arithmetic units run in parallel. The atomic unit of activity is called *action* consisting of large blocks of data and a set of commands to process that data.

The reminder of this whitepaper discusses the OpenSPL features on the arithmetic level (Section II), the architecture level (Section III) and system level (Section IV). We also provide some examples of spatial programs in Section V. We refer the interested reader to the OpenSPL Standard [7] for all additional details.

II. Spatial Arithmetic

A key property of spatial computing is that each operation can be instantiated as an actual physically separate arithmetic unit. As such, each of these units can implement different arithmetic and in fact a customized number representation as long as numbers get transformed correctly as they are forwarded from one unit to the next. For example, one spatial program could have 3-bit integers, 17-bit floating point and 27-bit logarithmic numbers, to provide an optimal solution. By optimizing the representation we are not only maximizing the number of arithmetic units we can fit on a chip, but also equally maximizing data transfer rates in and out of memories and across the I/O network.

At the bit-level, OpenSPL programs can fully exploit arithmetic optimizations such as, for example, minimizing the number of '1's in binary numbers, as shown most eloquently in [1] via sparse-coefficient polynomial approximation. In the spatial computing systems such optimizations result in linear savings of both space and power consumption since the zeros are omitted in the algorithm implementation.

In higher level arithmetic such as matrix algebra, the representation of matrix elements is important [3], particularly the location of the non-zero elements in sparse matrix computations [4]. By encoding entire data structures in a spatial computing manner, one can reduce the amount of data moving between memory and computational units and therefore achieve unprecedented performance and efficiency improvements. With temporal computing encoding and decoding would take time and eventually cancel out all of the advantages. In spatial computing, encoding and decoding just consume a bit of additional space.

III. Spatial Data Choreography

Once the arithmetic is laid out in space, the data needs to somehow flow through spatial "sub-programs", memory, and IO networks. This entire process needs to be scheduled in order to make sure no unit has to wait for another to finish and all of them operate all of the time. Optimal scheduling can be achieved by considering all three basic ingredients arithmetic, memory and networks in a holistic way. Spatial computer systems allow programmers to benefit from different tradeoffs, e.g., space for bandwidth or memory size for arithmetic complexity to name few. In this process the main objective is to optimize data access by reducing its size (encoding or variable precision representation) and minimize movement (improved reuse). Since the data is the "oil" that makes spatial computing operate smoothly, an OpenSPL compiler has to balance the movement of data to ensure that any two numbers meet at the right time and the right location in space. Of course such global matchmaking will differ from one SCS to another.

There is a conceptual difference in execution philosophy between temporal and spatial computing that can be described as "Hurry and wait versus Just-in-Time computing" [6]. While all of the contemporary high-performance systems care about finishing one task in the shortest time possible, in spatial computing we consider all of the tasks at once and balance the system in such a way that all interconnected components (e.g., arithmetic pipelines, memories and network links) are "impedance matched" and steadily used to maximize transfer efficiency between the different stages inside an SCS.

IV. Spatial Benchmarking

By laying out the arithmetic organization in space, and by flowing the data through the network, spatial computing systems are able to generate one result during every unit of time. In conventional CMOS technology energy is being consumed (and mostly wasted) at every clock signal tick. The efficiency of a spatial computing system is determined by how many ticks are needed in order to produce every single result. Turning the discussion upside-down, to achieve a certain wall clock execution time, spatial computers clock frequency can safely be orders of magnitude slower than in the case of temporal computing. This directly reduces the physical space required due to the smaller heatsinks and the relaxed cooling demands.

Benchmarking spatial computing systems is not trivial. In a paper from the Communications of the ACM [2] the dilemma and challenges facing the traditional benchmarking model are discussed. Spatial computing is called to answer the challenges posed by large software systems with immense datavolumes (Big Data, both structured and unstructured), or practically infinite streams of data. Thus, comparisons need to be made on complete, large-scale applications, with realistic large datasets. The comparison metrics stressing the benefits of spatial computing are: computations per cubic foot of datacenter space, computations per Watt, and operational costs per computation.

V. OpenSPL Examples

We examine two motivating examples of OpenSPL programs. The `SCS` string stands for the Spatial Computing Substrate as explained earlier and could be replaced with `DFE` for a dataflow engine substrate or any other short string representing a suitable special- and general purpose computing substrate which lend themselves to spatial computing instantiation, for example `QPU` could be used if someone were to create a spatial quantum computing substrate. Selecting custom SCS strings makes the substrate and custom optimizations explicit. It should be clear that `DFEVar` is quite different from `QPUVar` and hence will require different algorithmic optimizations to achieve the main goals of computing in space, namely reducing energy consumption per useful computation and maximizing the amount of computation per cubic foot. In this OpenSPL is unique in not creating false expectations of code portability since the same application driven challenges will be solved differently on different SCS instantiations. At the same time OpenSPL empowers programmers with all the necessary means to express powerful spatial optimizations and achieve the most efficient computations for their algorithms on a specific SCS.

The first motivating example depicted in Figure 1 leads to the instantiation of both computational structures in space that compute both values at each clock tick while the correct

```
class SimpleKernel extends Kernel {
    SimpleKernel() {
        SCSVar x = io.input("x", scsFix(24));
        SCSVar result = (x > 10) ? x+1 : x-1;
        io.output("y", result, scsFix(25));
    }
}
```

Figure 1. Simple Controlflow in Dataflow Kernel example. Both candidates (x+1 and x-1) are simultaneously computed in space and only the correct value flows out via the SCS Var "result.

```
class MovingAvgSimpleKernel extends Kernel {
    MovingAvgSimpleKernel() {
        SCSVar x = io.input("x", scsFloat(7, 17));
        SCSVar prev = stream.offset(x, -1);
        SCSVar next = stream.offset(x, 1);
        SCSVar sum = prev + x + next;
        SCSVar result = sum/3;
        io.output("y", result, scsFloat(7, 17));
    }
}
```

Figure 2. Moving Average Kernel example. Application specific floating point precision numbers with 7 bit exponent and 17 bit mantissa are used.

value is selected by a multiplexer controlled by the value of the select condition $(x > 10)$.

Another example that demonstrates the OpenSPL expressiveness is the moving average kernel shown in Figure 2. In this example *stream offsets* enable access of values from the stream relative to the current position. To support this, the substrate instantiation can define buffers, for example, in FMem space. In a typical SCS instantiation 10s of thousands of stream points can typically be buffered. Customized (floating point) representation as used in this example, is a key competitive advantage of computing in 2D space. Compared to the widely used IEEE single precision standard (8 / 24 bits), this example shows a powerful application specific optimization.

The automatically generated kernel graphs of both examples are shown in Figure 3. They represent the spatial arithmetic as it will be implemented on the targeted SCS. In the kernel graph on the right also the internal intermediate values are depicted to exemplify the working of the moving average filter on a simple stream of values. The stream directions are shown by the two arrows on the top and the bottom. As it can be seen from the graphs, the structures generated are typically simple, hence a significant number can be implemented on the SCS at the same time. This shows the spatial computing systems' advantage in terms of compute density as compared to all traditional multi- and many core platforms.

VI. Conclusion

Spatial Computing has been around for many decades in various shapes and forms. OpenSPL creates a platform for research, development and commercial offerings, uniting domain experts, scientists and application programmers with engineers and computer scientists to solve our future computational challenges and go beyond the limitations inherent

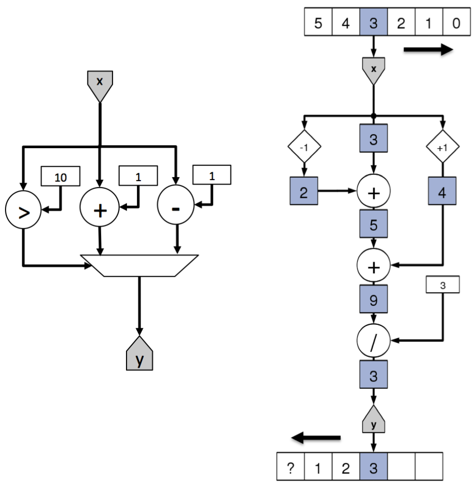

Figure 3. The Kernel graphs of the two examples. All coefficients (10, 1 and 1 on the left and 3 on the right) can be set externally using the SCS IO interface. In the right graph integer data values are used instead of the example's custom floating point (7, 17) for the sake of simplicity.

to multiprocessor and multicore computing systems and provide an answer for many of the challenges driven by the convergence trends in industry [5].

References

[1] N. Brisebarre, J. M. Muller and A. Tisserand, *Sparse-coefficient polynomial approximations for hardware implementations*, In Proc. of 38th Asilomar Conference on Signals, Systems and Computers, pp 532-535, California, USA, 2004.

[2] Michael J. Flynn, Oskar Mencer, Veljko Milutinovic, Goran Rakocevic, Per Stenstrom, Roman Trobec, Mateo Valero, *Moving from Petaflops to Petadata*, Communications of the ACM, Vol. 56 No. 5, pp 39-42, May 2013.

[3] C. B. Ciobanu, G. N. Gaydadjiev, *Separable 2D Convolution with Polymorphic Register Files*, ARCS 2013, pp 317-328.

[4] O. Lindtjorn, R. G. Clapp, O. Pell, O. Mencer, M. J. Flynn and H. Fu, *Beyond Traditional Microprocessors for Geoscience High-Performance Computing Applications*, IEEE Micro, vol. 31, no. 2, March/April 2011.

[5] O. Mencer, *Trading & Risk: What's Your Favorite Risk Flavor?*, Wall Street and Technology, http://www.wallstreetandtech.com/financial-risk-management/trading-risk-whats-your-favorite-risk-f/240164284

[6] M. Morf, *personal communication*, Stanford, USA.

[7] OpenSPL Consortium, *The OpenSPL Standard, v1.0*, http://www.openspl.org/

[8] J. E. Smith, *Decoupled access/execute computer architectures*, Computer Systems, ACM Transactions on; Volume 2, Issue 4, pp 289-308, November 1984.