Computational Spacetimes

E. Theodore L. Omtzigt
Geometric Computer Systems

Abstract

The execution of an algorithm is limited by physical constraints rooted in the finite speed of signal propagation. To optimize the usage of the physical degrees of freedom provided by a computational engine, one must apply all relevant technological and physical constraints to the temporal and spatial structure of a computational procedure. Computational spacetimes make explicit both technological an physical constraints, and facilitates reasoning about the relative efficiency of parallel algorithms through explicit physical complexity measures. Similar to Minkowski spacetime being the world model for physical events, computational spacetimes are the world model for computational events. Algorithms are specified in a spatial single assignment form, which makes all assignments spatially explicit. The computational spacetime and the spatial single assignment form provide the framework for the design, analysis, and execution of fine-grain parallel algorithms.

Introduction

Traditionally, models of computation have concentrated on modelling the *complexity* of algorithms expressed in terms of either the number of operations or the amount of memory required. The classification of algorithms into complexity classes provides insight into the structure of computational problems, and how 'hard' they are to solve. However, there is a difference between the analysis of the time and space complexity of an algorithm, and the analysis of the efficiency of execution. The complexity of an algorithm under a simplified model such as the random access machine provides a guideline in the form of upper and lower bounds. Evaluating the running time of an algorithm requires a more detailed analysis in which the constraints on execution are made explicit.

An algorithm specifies a computational procedure. It is finite in its description and the procedure consists of discrete steps that can be carried out mechanically. The Church-Turing thesis states that an algorithm can be expressed by a Turing machine, and nothing will be considered an algorithm if it cannot be presented as a Turing machine. It speaks for the completeness of the Turing machine model that, implicit to its operation, it does not violate the constraints of spacetime. However, the Turing machine as a physical system can be improved upon with respect to computational efficiency through parallelism.

The construction of computational machines inevitably must rely on technology. Currently, to build either digital or analog computing machines designers almost exclusively use integrated circuit technology. The circuit techniques used to implement binary logic translate the degrees of freedom provided by electrons and semiconductor materials into reliable and robust computing devices. This translation trades off robustness (or repeatability) with speed of operation. For the discussion here it is also important to notice that this translation creates very different resources and interactions for a computational procedure than the resources and interactions provided by nature. This leads to the observation that until we are capable of using the principle degrees of freedom inherent to the fundamental laws of physics, technology will be a factor in both the structure of algorithms as well as the efficiency of their execution.

The efficacy of an algorithm is contingent on the structure of the machine it is executed on. In a world with limited resources, the intricate interaction between algorithm and physical machine modulates the utilization of resources. In the absence of contention, resource utilization depends on the ratio between computation and communication. Counting operations to estimate the execution time of an algorithm is sufficient only if the computation to communication ratio is much larger than one. Since communication delays arise from both physical distances and the properties of materials, measures that reflect the constraints that nature and technology force on execution are required to design algorithms that execute efficiently. Otherwise stated: algorithms should be interpreted as physical systems if the computation delay of a basic arithmetic operation is much smaller than the delay of communicating the output of one operation to the input of another.

This paper introduces a model, called a *computational spacetime*, which makes the temporal and spatial constraints of a computational engine explicit. To specify computation and to take physical distances into account, the algorithm is specified in a spatial single assignment

0-8186-6715-X/94 $4.00 © 1994 IEEE

form. A single assignment form is a specification that makes all assignments in an algorithm explicit. The computational dependencies between the single assignment statements define a partial order on the computation. This partial order must match the geometry of the computational spacetime for the algorithm to execute efficiently on the computational engine. The computational spacetime and the spatial single assignment form provide the framework for the design, analysis, and execution of fine-grain parallelism.

Given the computational speed (100ps–1ns) and the propagation delays (1–10ns) of current (1990) technology it is evident that to make efficient use of computational resources, localized interconnections between functional units must be used. Any of the thirty two crystallographic point groups of three-space are possible candidates of parallel computing structures, however, the description of activity in lattices other than the cubic lattice, are complicated. By carefully selecting the underlying machine architecture, we can operate with only spatial structure; in the simplest case the integer lattice Z^n. Computation graphs, which make all computations within an algorithm explicit, are embedded within such lattices.

The computational spacetime model was created to argue from first principles that localized interconnections are required for parallel machines with fast arithmetic units. The next section will define the computational spacetime model of parallel computation. The model will be used to define the concepts of tightly and loosely coupled parallelism. The section on single assignment algorithms describes the spatial extensions to the single assignment form to be able to specify spatial relations among computations. The fourth section introduces the framework for the design, analysis, and execution of tightly coupled parallel algorithms. Finally, the last section summarizes the conclusions.

Computational Spacetimes

The efficiency of physical systems is proportional to the utilization of their resources. Some amount of work W is required to solve a computational problem. If the machine consists of p resources capable of one unit of work per unit of time, then the efficiency of the machine is $\frac{W}{pT}$ where T is the total number of time units the machine is occupied. One hundred percent resource utilization is not sufficient to ensure an effective computational procedure; the total amount of energy used to accomplish a computational task must be minimized.

Relaxing this energy argument, assume that energy requirements are proportional to time, then given an individual resource, its computational efficiency is

$$\frac{\Delta_{comp}}{\Delta_{comp} + \Delta_{com}},$$

where Δ_{comp} is the time required to complete a computation, and Δ_{com} is the time required to communicate the inputs. Given a finite time, the neighborhood from which a functional unit can receive its inputs is limited. In current technologies the delays of logic and simple arithmetic operations are so short that the neighborhoods are confined to the chip die to obtain a reasonable efficiency of the functional unit. In the case of tightly-coupled, fine-grain parallelism, the spatial organization of many simple operations is essential to efficient execution, increasingly so when technology provides faster active elements and higher integration densities. In the following, the above discussion is made more precise.

Spacetime provides physics with a model of the world. The most basic idea of spacetime is that of an *event*. The events of spacetime are considered 'point' occurrences, that is, events have no spatial or temporal extent. In free space, the *distance* between two neighboring events is given by the *differential squared interval*

$$ds^2 = v^2 dt^2 - dx^2 - dy^2 - dz^2 \qquad (1.1)$$

where v is the maximum propagation speed of a signal. The differential squared interval is a description of a spherical wavefront in three space, or a circular wavefront in two space. This wavefront is called the *null cone*. The null cones, and the respective displacement vectors, define an order on the physical events that take place.

In Minkowski spacetime, the speed of light c is used in the differential squared interval, and the cones are called light cones. The manifold described by Minkowski spacetime is richer than is required for a model of parallel computation. For example, time and space dilations due to relativity are inconsequential for computation on a *stationary* grid of processors. Important for the discussion here is the separation of events in different event sets due to the limit on signal propagation.

To be able to communicate information between two vertices of a stationary lattice, the world line corresponding to the receiving vertex must cut the null cone that corresponds to the vertex at which the event originated. Since the cones are ever-increasing, events that originate at the lattice points of a stationary lattice will always be able to influence another site, simply by allowing time to pass. But is this time available? The efficiency of a computational resource drops as the time spent communicating the operands increases with respect to the time spent executing. Rewriting the equation describing the efficiency of a computational resource yields

$$\eta = \frac{\Delta_{comp}}{\Delta_{comp} + \Delta_{com}} ,$$

$$\Delta_{com} = \frac{1 - \eta}{\eta} \cdot \Delta_{comp} , \qquad (1.2)$$

where η denotes the efficiency of a computational resource, Δ_{comp} is the delay of an operation of the functional unit, and Δ_{com} is the delay to communicate the operands.

Spacetime couples space and time; some physical neighborhood, described by the metric of spacetime, corresponds to some elapsed time. This relationship can be used to define a spatial extent as a function of a desired efficiency of the functional units. The *event neighborhood*, δ, is defined as

$$\Delta s(t) = v_{propagation} \cdot t - f(x,y,z),$$

$$\begin{aligned} \delta(\Delta_{comp},\eta) &\equiv f(x,y,z) \\ &= \left(\frac{1 - \eta}{\eta}\right) \cdot \Delta_{comp} \cdot v_{propagation} \ . \qquad (1.3) \end{aligned}$$

Equation (1.3) effectively truncates the null cones, causing events that lie within the absolute future of a particular event to appear spacelike if they cannot be reached in the time given by $\frac{1 - \eta}{\eta} \cdot \Delta_{comp}$. Equation (1.3) uses a generic description of the physical neighborhood. Although physical events take place in free space, and physical change can be communicated on the wavefront of an expanding sphere, the communication of an event within a computational structure tends to have an additional constraint: the interconnection network.

Computation can be represented by a directed graph in which nodes represent operations, and arcs denote precedences (or dependences) among the operations. The computation graph makes all operations and their interaction explicit. A machine can be represented by a graph as well; the nodes represent functional units, and the arcs represent communication channels between the functional units. To execute the computation on a machine, the computation graph must be embedded in the machine graph. The data precedence relations in the computation graph are confined to the communication channels of the machine graph, which changes the metric of the space in which the computation takes place. A model that characterizes the temporal and spatial constraints is called a *computational spacetime.*

A *computational event* in a computational spacetime is a basic operation, such as an add or multiply, or a logic AND operation, and is considered a point-occurrence. Computational events can only take place at computational resources. In a physical system, these resources must be mounted. This rigging is modelled by a stationary lattice. At each lattice point, one and only one computational resource, or *functional unit*, can be found. Since a functional unit is an object having continuous existence and its position with respect to the other functional units remains constant, it is represented in a spacetime diagram by a straight world line.

A *communication event* is a communication of one data element from one functional unit to a direct neighbor. *Direct neighbors* are functional units that share a physical communication link. Since data is produced and consumed, a data element spans only a segment of a world line in the spacetime diagram. To effectively specify a parallel algorithm, a designer is interested in the global properties of the communication events within the algorithm and the underlying computing machine. The communication between functional units occurs along some interconnection network. The connectivity of the interconnection network determines the geometry of the space in which communication events take place. This *geometry* describes the computational spacetime. For example, a cubic interconnection network can be described by a cubic lattice with the metric:

$$\Delta s_{cubic} = v\Delta t - (|\Delta x| + |\Delta y| + |\Delta z|) \qquad (1.4)$$

In this case, the null cone is an expanding cube. The null cone of a two dimensional equivalent is given in Figure 1.1.

The *intrinsic time step* of a computational spacetime is defined as the smallest time step in the system. This time step is used to truncate the null cones to ensure a reasonable efficiency of the functional units in the machine. A spacetime diagram of a stationary lattice is a collection of truncated null cones, or *compartments.* Within such a compartment only a fixed number of *computational activities* can take place, where a computational activity is defined as a basic operation plus a communication across one and only one link connecting two vertices. The compartments are an abstraction of the limit on the speed of signal propagation and the connectivity of the interconnection network.

The event neighborhoods allow us to precisely define the otherwise vague concepts of tightly and loosely-coupled parallelism. Assume a computational spacetime with an event neighborhood $\delta(t)$, and two communicating processes, *u* and *v*. The length of a computational event is given by Δ_{comp}. Assume *u* and *v* have some spatial separation Δ_{uv}. Using equation (1.3), the two processes are called tight-

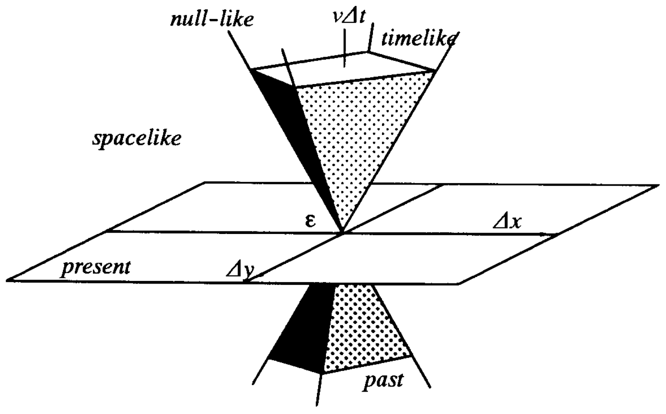

Figure 1.1 The null cone of a square lattice defined by the metric of equation (1.4)

ly-coupled if and only if

$$\Delta_{xy} < \delta(\Delta_{comp}, \eta) , \qquad (1.5)$$

for some efficiency η. Furthermore, assume an aggregate of such processes distributed through space. This collection is called tightly-coupled if and only if for all distances, Δ_{uv}, between communicating processes, equation (1.5) holds. Any other collection is called loosely-coupled. Equation (1.5) shows that any computation can be made tightly-coupled by reducing the efficiency of the computational resource.

Single assignment algorithms

Systolic algorithms are tightly-coupled fine-grain algorithms in which spatial movement of data is specified for each time step of the algorithm. However, two problems are associated with systolic algorithms. First, in a systolic algorithm both space and time are explicit. Since both space and time are part of the functionality, systolic algorithms become unmanageable quickly. Second, the lock-step control mechanism of a systolic algorithm does not address the sequencing of independent operations that require service from the same functional unit in the same clock tick. The lack of a control mechanism to share resources makes systolic algorithms useful only for dedicated architectures, but these architectures can be very fast, explaining their popularity in signal processing applications.

Now consider a single assignment algorithm. Since a single assignment algorithm has no notion of space or time, its functionality is much easier to design than the corresponding systolic algorithm. When the computation graph is embedded in some physical structure, out of the inherent composition of the algorithm activity will evolve, modulated by the laws of physics, resource constraints, and the timing of the inputs. The association of spatial and temporal dimensions to the computation graph can be postponed until run-time, which makes single assignment algorithms perfectly suited to specify massively parallel algorithms.

In the previous section we have seen that efficient parallel computation requires short physical distances between dependent computations. A sequential language is defined with respect to a featureless memory in which spatial relations are 'flattened'; no notion exists of a location dependent time difference between memory cells. An abstraction to specify spatial relations between computations must be added. Since in a single assignment algorithm each and every operation is made explicit, the computation graph is explicit in the program text. Spatial relations can be added by embedding the computation graph in a lattice with a distance measure.

In a single assignment algorithm, a unique name for the result of each operation is required. Assuming the lattice Z^N, the physical embedding of the computation graph and the assignment of unique names can be combined by associating an index to a variable. This index is interpreted as the lattice point at which this operation takes place. For example, an algorithm for vector addition might embed the vectors in a one-dimensional lattice Z^1, and assign each individual addition to a lattice point,

```
{(i) | 1 ≤ i ≤ N}   {
        x_i  =  y_i  +  z_i
}
```

This results in three data items per lattice point plus a functional unit capable of performing the add. The domain $\{(i) \mid 1 \le i \le N\}$ is called the index domain of the algorithm.

This type of regular algorithm, ubiquitous in parallel computation, can be described by a set of equations F_i defined over convex domains D_i and lattices L_i

$$\text{algorithm} = \{(F_i, D_i, L_i) \mid 1 \leq i \leq f\},$$

where f denotes the number of equations in the algorithm. A computation is specified as a dependence relation, which is a map β from an index point p to another index point $p\beta$,

$$p\beta = pB + b ,$$

where B is the linear part of the dependence map, and b is the affine part. For example, in the following single assignment statement, the functionality is a multiply-add

$$x_{i,j,k} = x_{i,j,k-1} + y_{i,j,k} \cdot z_{i,j,k} ,$$

with dependence maps given in equation (1.6). These particular dependence maps are *uniform*, that is, the linear part of the map is the identity matrix which implies that the mapping is independent of the position of invocation. If the linear part is not the identity matrix then the transformation is an *affine dependence map.*

By definition, the left-hand-side of an equation has standard indices. Each equation F_i defines a recurrence variable, or *rvar*. In the above example, the recurrence variable is x. The variables on the right-hand-side of the equation are recurrence variables as well, but they are defined elsewhere. The recurrence variable represents a collection of communicating processes; it is an aggregate object with a well-defined extent given by D_i and L_i. D_i is called the domain of computation of the recurrence variable, and L_i the lattice of computation.

Low level specifications of algorithms that take physical constraints into account can be found in [Hennie, 1961, 1968]. Systolic algorithms [Kung, 1982][Leiserson, Saxe, 1984] are similar to the systems studied by Hennie, and have been extended by a number of authors, most notably [Quinton, 1983], [Moldovan, 1983], [Cappello and Steiglitz, 1983], [Delosme and Ipsen, 1986], and [Mauras, et al., 1990]. The earliest study into the formal characteristics of recurrence equations can be found in [Karp and Miller, 1966], and [Karp, Miller, Winograd, 1967]. A complete framework for the design, analysis, and execution of affine extensions to uniform recurrence equations studied by Karp, Miller, and Winograd, is presented in [Omtzigt, 1992].

A framework for the design, analysis, and execution of fine-grain parallel algorithms

Every computing machine creates some computational spacetime, described by the lattice of processors and an event neighborhood. The interconnection structure between the processors determines the metric of the computational spacetime. Combined with the intrinsic time step, provided by the shortest computational delay in the system, this metric defines the shape and extent of the event neighborhood.

If this procedure is applied to existing parallel architectures one realizes that to determine the computational spacetime created by a machine is not that easy. Interconnection networks have rarely been designed to be uniform with respect to three-space, resulting in complicated neighborhood functions. Moreover, global interconnection networks are laid out across different technologies in such a way that improvements in technology change the characteristics of the neighborhood functions significantly. This changes the trade-offs between computation and communication causing the efficiency of an algorithm to be dependent on the size of the machine and technology used to implement the machine.

The description of a computational procedure typically involves three levels; the overall structure of the algorithm, the individual arithmetic and/or logic operations, and the bit-level implementation of the basic operations. All these levels require control, which itself is a form of computation. Depending on the amount of resource sharing required, specification of control varies. In a sequential machine, almost all code specifies control, whereas for the single assignment algorithms almost all code specifies computational structure. The problem of this state of affairs is that it makes an algorithm very dependent on the implementation of the machine. This dependency complicates the automation of moving programs from one machine architecture to another.

It is easier to design efficient algorithms, or to construct an optimizing compiler, for uniform architectures because the interactions between constraints are simplified. If the interconnection network of the machine is symmetric with respect to three-space, and the event neighborhood is chosen such that the architecture can support the same com-

$$\begin{array}{llll} x \rightarrow x & p\beta = pB + b = p\mathrm{I} + \begin{bmatrix} 0 & 0 & -1 \end{bmatrix} & & \\ x \rightarrow y & p\beta = pB + b = p\mathrm{I} + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} & & (1.6) \\ x \rightarrow z & p\beta = pB + b = p\mathrm{I} + \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} . & & \end{array}$$

putation and communication ratio across several generations of technology then the basic operation of the machine can be held constant. In this case, an algorithm can be designed independent of the size and implementation technology of the machine.

Computational crystals

The separation of events due to fast state transitions of high-speed devices and a limit on signal propagation, complemented by the limited wire density of electronic technologies, makes localized architectures faster, more efficient, easier to compile for, and easier to manufacture than non-localized architectures. Nature provides many examples of localized configurations; every solid is a localized configuration of atoms. In three-space there are thirty-two different crystal lattices, such as cubic, tetragonal or ortho-rhombic lattices. By arranging computational resources along the structure of some crystal lattice, a computational structure is created where the communication links between the 'atoms' are localized, and the computation-to-communication ratio is independent of the number of atoms. At the same time, such a computational crystal allows enough flexibility in the implementation of the machine to keep the ratio constant across several generations of integrated circuit technology. The crystal lattices form crystallographic point groups, and the generators of these groups can be used to create routes between the vertices.

For the development of a method to execute a computation graph on a fixed-size architecture, the cubic lattice is most suited. If the event neighborhood contains only direct neighbors then the spacetime diagram of a three-dimensional orthogonal lattice becomes a four-dimensional lattice of compartments that have their vertices on the lattice points of Z^4. Similarly, lower dimensional orthogonal lattices are subsets of the cubic lattice. Higher dimensional lattices can be mapped onto lower dimensional lattices through projections along the axes. The computation to communication ratio of the lower dimensional lattice is identical to the higher dimensional lattice under the assumption that the control mechanism keeps physical distances within some bound.

However, the computational activity of a parallel algorithm projected along the axes of Euclidean space may exhibit inefficiencies due to collisions of activity. A direction of projection other than along the axes may avoid such collisions, and should not be excluded. To remain in Z^n oblique projections must be used. The relaxation of using oblique projections can cause local communication patterns to become non-local, creating possible resource contention that was not modelled in the original algorithm. The spatial reductions must thus be chosen with care.

This suggests the following method to design, analyze, and execute algorithms with tightly-coupled, fine-grain parallelism.

1. Assume an orthogonal lattice Z^n and equate this with the machine graph. The communication links are given by the basic vectors of the lattice,
2. embed the computation graph in this lattice, allowing only a fixed number of computational events at a single vertex,
3. rewrite communication vectors that span multiple event neighborhoods to be collections of single event neighborhoods,
4. select projection directions along which to reduce the dimensionality of the computation graph without causing excessive activity clashes and unnecessary dilation of communication vectors,
5. execute the projected computation graph on some machine (preferable a machine that produces the lattice Z^n as vertices in the computational spacetime it embodies).

Conclusions

Fine-grain parallelism requires carefully designed spatial relations to execute efficiently. A computational spacetime models the spatial structure and temporal constraints that influence the execution of an algorithm. An event neighborhood is defined as the spatial extent a computational event can influence during some intrinsic time step. By embedding the computation graph, which makes all computation and communication explicit, into a computational spacetime, spatial and temporal constraints are accounted for. However, this embedding requires the designer to distinguish between space and time, desirably deferred until run-time. The properties of an orthogonal lattice are instrumental in providing this relaxation.

The computational spacetime of an orthogonal lattice can be made uniform in space and time by choosing the intrinsic time step such that an event neighborhood contains only direct neighbors. The vertices of the truncated null cones fall on the lattice points of an n+1 dimensional Euclidean space Z^{n+1}, for $n \leq 3$. Otherwise stated, under the stipulation that dependent computations must be placed at direct neighbors, all computational resources and their interconnectivity are made explicit by embedding the computation graph in Z^n, for $n \leq 3$.

The physics of computation combined with a dose of technical realism has an opportunity to alleviate the fundamental problem of machine dependency of algorithms. Except for the work in systolic algorithms and cellular automata, the current state of algorithm design ignores physics at the conceptual level. The approach followed in systolic algorithms and cellular automata, in increasing order of religion, is that the physics defines a machine, which defines an algorithm. Ultimately, when we are able to use the degrees of freedom available in nature directly, the physics determines the algorithm without the need of some computational engine which abstracts these fundamental degrees of freedom into something less efficient, but closer to the conceptual computation step of the Turing or random access machine.

References

[1] Cappello, P.R. and K. Steiglitz, "Unifying VLSI Array Designs with Geometric Transformations," *Proceedings of the International Conference on Parallel Processing*, 1983, pp. 448-457.

[2] Delosme, Jean-Marc and Ilse Ipsen, "Systolic Array Synthesis: Computability and Time Cones," *Parallel Algorithms and Architectures*, M. Cosnard *et al.*, (editors), Elsevier Science Publishers, 1986, pp. 295-312.

[3] Hennie, F.C., *Iterative Arrays of Logical Circuits*, The MIT Press and John Wiley & Sons, Inc., 1961.

[4] Hennie, F.C., *Finite State Models for Logical Machines*, John Wiley & Sons, Inc., 1968.

[5] Karp, Richard and Raymond Miller, "Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing," *SIAM Journal of Applied Mathematics*, vol. 14, no. 6, 1966, pp. 1390-1411.

[6] Karp, Richard, Raymond Miller and Shamuel Winograd, "The Organization of Computations for Uniform Recurrence Equations," *Journal of the Association of Computing Machinery*, vol. 14, no. 1, 1967, pp. 563-590.

[7] Kung, H.T., "Why Systolic Architectures," *Computer Magazine*, vol. 15, no. 1, January 1982

[8] Leiserson, Charles E. and James B. Saxe, "Optimizing Synchronous Systems," *Journal of VLSI and Computer Systems*, vol. 1, no. 1, 1984.

[9] Mauras, C., P. Quinton, S. Rajopadhye and Y. Saouter, "Scheduling Affine Parameterized Recurrences by means of Variable Dependent Timing Functions," *Proceedings of the International Conference on Application Specific Array Processors*, IEEE Computer Society, 1990.

[10] Moldovan, Dan I., "On the Analysis and Synthesis of VLSI Algorithms," *IEEE Transactions on Computers*, vol. c-31, no. 11, November 1982, pp. 1121-1126.

[11] Moldovan, Dan I., "On the Design of Algorithms for VLSI Systolic Arrays," *Proceedings of the IEEE*, vol. 71, no. 1, January 1983, pp. 113-120.

[12] Omtzigt, E. Theodore L., "Domain Flow and Streaming Architectures," in *Proc. of the Int'l Conference on Application Specific Array Processors*, IEEE Computer Society, 1990

[13] Omtzigt, E. Theodore L., *Domain Flow and Streaming Architectures: A paradigm for efficient parallel computing*, Ph.D. Dissertation, Yale University, 1992

[14] Quinton, Patrice, "The Systematic Design of Systolic Arrays," IRISA Research Report, no. 193, March 1983.