A Lattice Model for the Optimization of Communication in Parallel Algorithms

Sven DE SMET [a,1],
[a] *Student at Ghent University*

Abstract. This paper describes a unified model for the optimization of communication in parallel algorithms and architectures. Based on a property that provides a unified view of locality in space and time, an algorithm is constructed that generates a parallel architecture that is optimized for communication for a given computation. The optimization algorithm is constructed using the lattice algebraic properties of congruence relations and is therefor applicable in a general context. An application to a bio-informatics algorithm demonstrates the value of the model and optimization algorithm.

Keywords. Parallelism, Communication, Locality, Architecture

Introduction

In order to optimally benefit from the incessant increase in parallel computational capacity in modern architectures, the available parallelism in our algorithms must increase accordingly. It is well-known that communication in parallel architectures creates a bottleneck that limits the benefits of parallelism in many cases. Since manual optimization of communication in parallel architectures requires considerable effort, we must resort to automatic optimization.

This paper describes an abstract model of a parallel, computing machine and its communication architecture. The model further defines the abstract property **communication regularity** that represents an opportunity to optimize communication while constructing an architecture for a specific algorithm. This property unifies **data distribution** [1,2] and **array contraction** [3,4] in their most general form. An algorithm to construct an architecture with optimized communication for a given computation is given. The optimization algorithm is demonstrated by applying it to a bio-informatics algorithm in section 6.

1. Mathematical Background

1.1. Notation

Let $a..b$ with $a, b \in \mathbb{Z}$ denote the set of integers from a up to and including b: $a..b \triangleq \{i | i \in \mathbb{Z} \wedge a \leq i \leq b\}$. For a vector or tuple x, let $|x|$ denote the number of elements it contains.

[1] sven.desmet@cubiccarrot.com, http://www.cubiccarrot.com/salmoc/

For a set A, let $\mathcal{P}A$ denote the set of all its subsets and $\mathcal{L}A \triangleq \mathcal{P}(A^2)$ denote the set of binary relations on A. The **Kleene Closure** of A is denoted with $A^* \triangleq \bigcup_i^{\mathbb{N}} A^i$.

For a relation $\lessdot$ on $\mathbb{A}$ and a set $\mathbb{B} \subseteq \mathbb{A}$, let $\mathcal{U}^{\mathbb{A}}_{\lessdot}\mathbb{B}$ denote the **earliest $\lessdot$-successor** of $\mathbb{B}$ on $\mathbb{A}$, which is defined by the axiom

$$\forall_{\mathbb{B}}^{\mathcal{P}\mathbb{A}} \mathcal{U}^{\mathbb{A}}_{\lessdot}\mathbb{B} \in \mathbb{A} \wedge \forall_b^{\mathbb{B}} b \lessdot \mathcal{U}^{\mathbb{A}}_{\lessdot}\mathbb{B} \wedge \forall_a^{\mathbb{A}}(\forall_b^{\mathbb{B}} b \lessdot a) \Rightarrow \mathcal{U}^{\mathbb{A}}_{\lessdot}\mathbb{B} \lessdot a \tag{1}$$

1.2. Partitions

1.2.1. Basic definitions

A **partition** $\mathbf{B}$ of a set $\mathbb{A}$ is a set of subsets of $\mathbb{A}$ such that their disjoint union is equal to $\mathbb{A}$, i.e. $\mathbb{A} \triangleq \bigsqcup_{\mathbb{B}}^{\mathbf{B}} \mathbb{B}$. The **cells** of a partition are the subsets of which it consists. Let $\mathcal{C}\mathbb{A}$ denote the set of all partitions of $\mathbb{A}$.

A partition $\mathbf{P}$ is **at least as fine as** a partition $\mathbf{Q}$, denoted as $\mathbf{P} \preceq \mathbf{Q}$ (or, $\mathbf{Q}$ is **at least as coarse as** $\mathbf{P}$, $\mathbf{Q} \succeq \mathbf{P}$), if every cell of $\mathbf{P}$ is contained in a cell of $\mathbf{Q}$:

$$\mathbf{P} \preceq \mathbf{Q} \triangleq \forall_P^{\mathbf{P}} \exists_Q^{\mathbf{Q}} P \subseteq Q \tag{2}$$

A partition $\mathbf{P}$ is **finer than** a partition $\mathbf{Q}$, denoted as $\mathbf{P} \prec \mathbf{Q}$ (or, $\mathbf{Q}$ is **coarser than** $\mathbf{P}$, $\mathbf{Q} \succ \mathbf{P}$) if $\mathbf{P} \preceq \mathbf{Q} \wedge \mathbf{P} \neq \mathbf{Q}$.

The **refining** operation is defined as $\otimes \triangleq \mathcal{U}^{\mathcal{C}\mathbb{A}}_{\succeq}$ and the **coarsening** operation is defined as $\odot \triangleq \mathcal{U}^{\mathcal{C}\mathbb{A}}_{\preceq}$. The **bottom** of $\mathcal{C}\mathbb{A}$ is $\bot_{\mathbb{A}} \triangleq \mathcal{U}^{\mathcal{C}\mathbb{A}}_{\succeq}\mathcal{C}\mathbb{A} = \{\{a\} | a \in \mathbb{A}\}$, the finest element, while the **top** of $\mathcal{C}\mathbb{A}$ is $\top_{\mathbb{A}} \triangleq \mathcal{U}^{\mathcal{C}\mathbb{A}}_{\preceq}\mathcal{C}\mathbb{A} = \{\mathbb{A}\}$, the coarsest element. For every set $\mathbb{A}$, the structure $(\mathcal{C}\mathbb{A}, \preceq, \otimes, \odot, \bot, \top)$ forms a **complete lattice**.

1.2.2. Constructions

Let $\mathbf{B}|_{\mathbb{A}} \triangleq \{B \cap \mathbb{A} | B \in \mathbf{B}\}$ denote the **restriction** of a partition $\mathbf{B} \in \mathcal{C}\mathbb{B}$ to a set $\mathbb{A} \subseteq \mathbb{B}$.

A **partition hierarchy** is a tuple of partitions where subsequent partitions in the tuple satisfy a binary relation. Let $\mathcal{H}_{\lessdot}A$ denote the set of partition hierarchies on A that satisfy the relation $\lessdot$,

$$\forall_A^{\mathcal{S}} \forall_{\lessdot}^{\mathcal{L}\mathcal{C}A} \mathcal{H}_{\lessdot}A \triangleq \{s | s \in (\mathcal{C}A)^* \wedge \forall_i^{2..|s|} s_{i-1} \lessdot s_i\} \tag{3}$$

Let $\mathcal{T}_{\lessdot}A$ with $\lessdot \in \mathcal{L}A$ denote the set of tree-structured graphs where the nodes are labeled with elements of A and the labels of parent and child-nodes satisfy the relation $\lessdot$,

$$\forall_A^{\mathcal{S}} \forall_{\lessdot}^{\mathcal{L}A} \mathcal{T}_{\lessdot}A \triangleq \{G | G \in \text{Trees}(A) \wedge \forall_{v,w}^{G^2} v \xrightarrow{G} w \Rightarrow v \lessdot w\} \tag{4}$$

where we have identified a tree $G \in \text{Trees}(A)$ with its set of nodes and we have $v \xrightarrow{G} w$ iff an edge $(v, w) \in G^2$ exists in the tree.

A **partition bijection** is a bijective relation between the cells of two partitions. A partition bijection on the sets C and D also specifies a partition of $C \cup D$ where each cell of the partition of $C \cup D$ is the union of a cell of the partition of C with its related cell of the partition of D.

A partition on a set $\mathbb{A}$ can be specified using a function $f \in \mathbb{A} \to \mathbb{B}$ to another set $\mathbb{B}$ by allocating all elements that are mapped to the same value by f to a cell unique to this value. Let us denote the resulting partition with $\mathbb{A}_{[f]}$,

$$\mathbb{A}_{[f]} \triangleq \{\{a | a \in \mathbb{A} \wedge f(a) = i\} | i \in \operatorname{Im} f\} \tag{5}$$

A set $Y \subseteq \mathcal{C}\mathbb{A}$ is an **abstraction** of the set of partitions on $\mathbb{A}$ iff $\otimes$ and $\odot$ are closed on $\mathbb{A}$ and $\{\bot_{\mathbb{A}}, \top_{\mathbb{A}}\} \subseteq Y$. The structure $(Y, \preceq, \otimes, \odot, \bot, \top)$ also forms a complete lattice.

2. Representation of Computations

A **computation** is a structure $(\Omega, \Gamma, \Delta, \rightsquigarrow)$ where

- Ω is the set of **operations** executed during the computation.
- Δ is the **data space**, the set of data elements accessed during the computation. Information can be stored in a data element and can later be retrieved from it. The data space is specified implicitly by Ω and Γ.
- $\Gamma \subseteq \Omega \to \Delta$ is the set of **access functions** that map an operation to a data element accessed by it. (In the full version of this paper, partial functions will be accounted for.)
- $\rightsquigarrow \in \mathcal{L}\Omega$ is a minimal partial order of the operations such that any execution of the computation must respect $\rightsquigarrow$ to obtain valid results

For the static specification and analysis of computations, Ω and Δ are often given as the finite union of sets of a specific type. In this case, the finite union of operations is indexed by $\mathcal{S}$, the set of **statements** , $\Omega \triangleq \bigcup_{s}^{\mathcal{S}} \Omega_s$, while the finite union of data elements is indexed by $\mathcal{V}$, the set of **variables**, $\Delta \triangleq \bigcup_{v}^{\mathcal{V}} \Delta_v$. Let $\Gamma_s^v \subseteq \Omega_s \to \Delta_v$ denote the access functions from $s \in \mathcal{S}$ to $v \in \mathcal{V}$.

3. An Abstract Architecture

Let $\mathbb{X} \triangleq \{\mathfrak{s}, \mathfrak{t}\}$ where $\mathfrak{s}$ denotes space and $\mathfrak{t}$ denotes time. Let $\overset{\circ}{\diamond}$ be an operation that transforms an element $x \in \mathbb{X}$ to the other element $\overset{\circ}{x} \in \mathbb{X} \setminus \{x\}$.

3.1. Specification

An **architecture** for a computation (Ω, Γ, Δ) is a structure $(\boldsymbol{\Omega}, \gamma, \boldsymbol{\Delta})$ where

- $\boldsymbol{\Omega}^{\diamond} \in \mathbb{X} \to \mathcal{H}_{\geq} \Omega$ is the **spacetime hierarchy**, which consists of

  - The **space hierarchy** $\mathbf{\Omega}^{\mathfrak{s}} \in \mathcal{H}_{\succeq}\Omega$. Every cell of every partition in this increasingly fine partition hierarchy can be identified with an abstract processing element on which the operations in the cell are executed. If the cell is further refined into subcells by a subsequent partition, the subcells of the cell can be identified with smaller parallel processing elements contained by it.
  - The **time hierarchy** $\mathbf{\Omega}^{\mathfrak{t}} \in \mathcal{H}_{\succeq}\Omega$. Every cell of every partition in $\mathbf{\Omega}^{\mathfrak{t}}$ can be identified with an abstract time interval during which the operations in the cell are executed. Subcells of this cell can be identified with smaller time intervals contained by it.

- $\mathbf{\Delta}^{\diamond}$ is the **communication topology**, which consists of

  - The **interconnection topology** $\mathbf{\Delta}^{\mathfrak{s}} \in \bigtimes_{i}^{1..|\mathbf{\Omega}^{\mathfrak{s}}|} \mathcal{T}_{\succeq}\mathcal{C}\mathbf{\Omega}^{\mathfrak{s}}_i$. Every cell of every partition in this tuple of partition trees can be identified with an abstract interconnection channel. An interconnection channel allows to transfer data between a set of processing elements. The subcells of an interconnection cell are identified with smaller interconnection channels that, combined, transfer the same data elements at a different level of the spacetime hierarchy. A partition tree has a level for every level of the time hierarchy.
  - The **memory topology** $\mathbf{\Delta}^{\mathfrak{t}} \in \bigtimes_{i}^{1..|\mathbf{\Omega}^{\mathfrak{t}}|} \mathcal{T}_{\succeq}\mathcal{C}\mathbf{\Omega}^{\mathfrak{t}}_i$. Every cell of every partition in $\mathbf{\Delta}^{\mathfrak{t}}$ can be identified with an abstract memory element. A memory element allows to store data during a specific time interval. Subcells of a memory cell are identified with smaller memory elements that are available during smaller intervals. A partition tree has a level for every level of the space hierarchy.

- The **access set topology** $\gamma \in \mathbb{X} \to (\mathcal{T}_{\supseteq}\mathcal{P}\Gamma)^*$. The structure of γ is identical to $\mathbf{\Delta}$, but every node of the trees is now labeled with a subset F of Γ. For $F \subseteq \Gamma$ the **operation-data partition** of $C \subseteq \Omega$ is the finest partition such that every operation is contained in the same cell as the data elements that it accesses:

$$\mathcal{B}_F(C) \triangleq \mathcal{U}_{\succeq}^{\mathcal{C}(C \cup \Delta)}\{\mathbf{Q} | \mathbf{Q} \in \mathcal{C}(C \cup \Delta) \wedge \bigwedge_{f}^{F} \bigwedge_{\omega,\delta}^{C\times\Delta} \delta = f(\omega) \Rightarrow \bigvee_{Q}^{\mathbf{Q}} \{\omega, \delta\} \subseteq Q\} \tag{6}$$

  For $F \subseteq \Gamma$ the **data-access partition** of $\mathbf{C} \in \mathcal{C}\Omega$ is

$$\mathcal{Q}_F(\mathbf{C}) \triangleq \bigcup_{C}^{\mathbf{C}} \mathcal{B}_F(C)|_{\Omega} \tag{7}$$

  For $x \in \mathbb{X} \wedge (g,h) \in 1..|\mathbf{\Omega}^x| \times 1..|\mathbf{\Omega}^{\mathring{x}}|$ the x-communication partition induced by an access set $F \subseteq \Gamma$ at the level of (g,h) is

$$\mathcal{K}^x_{g,h}(F) \triangleq (\mathbf{\Omega}^x_{g+1} \otimes \mathbf{\Omega}^{\mathring{x}}_h) \odot \mathcal{Q}_F(\mathbf{\Omega}^x_g \otimes \mathbf{\Omega}^{\mathring{x}}_h) \tag{8}$$

3.2. Classical Spacetime Partitioning and $\mathbf{\Omega}$

Notice that $\mathbf{\Omega}$ is more concrete than the abstract machine considered by Lim and Lam [5,6] in the context of classical affine spacetime partitioning in the sense that it

specifies the execution of the operations in more detail. Indeed, $\mathbf{\Omega}$ can be considered as a **simultaneous partitioning** of space and time which contrasts with the **alternating partitioning** that is used for finding a solution for synchronization-minimal parallelisation.

For parallelisation, a hierarchy $\boldsymbol{\phi} \in \mathcal{H}_{\succeq}\,\Omega$ is constructed where the partitions it contains are labeled as either space or time partitions. So, for a partition $\boldsymbol{\phi}_i$ in this tuple with $i \in 1..|\boldsymbol{\phi}|$, we have $\boldsymbol{\phi}_j \succeq \boldsymbol{\phi}_i$ for all $j \in 1..i$. A partition is therefore finer than both its finest preceding space and its finest preceding time partition.

In contrast, partitions in $\mathbf{\Omega}$ are only finer than the finest preceding partition of the same $\mathbb{X}$-type. The extra information encodes the relation between the subcells of cells that result from partitioning the operations w.r.t. to the other spacetime element. We can use this to optimize the communication hierarchy while extending the alternating partitioning that results from a parallelisation to a spacetime hierarchy $\mathbf{\Omega}$ with the same parallelism structure. The spacetime hierarchy models parallelism, scheduling and mapping in space and time in a unified way.

4. Architecture Optimization

4.1. Communication Regularity

For $x \in \mathbb{X} \wedge (g,h) \in 1..|\mathbf{\Omega}^x| \times 1..|\mathbf{\Omega}^{\mathring{x}}|$ there is x-**communication regularity** for an access set $F \subseteq \Gamma$ at the level of (g,h) iff $\mathcal{K}^x_{g,h}(F) \prec \mathbf{\Omega}^x_g \otimes \mathbf{\Omega}^{\mathring{x}}_h$.

If $x = \mathfrak{s}$ this allows to construct an interconnection partition $\mathcal{K}^x_{g,h}(F)$ where each common interconnection element contains less data elements and is accessed by less processing elements, leading to a generalization of **data distribution** which allows to use a smaller and faster interconnection element for each cell.

If $x = \mathfrak{t}$ this allows to construct a memory partition $\mathcal{K}^x_{g,h}(F)$ such that fewer data elements must be accessible during a shorter interval in each cell of the partition. If the intervals are totally ordered by the time partition, the same memory can be reused for the separate time cells. This results in a generalization of **array contraction** which allows to use a smaller and faster memory for each cell.

To optimize an architecture the simultaneous partitioning must be chosen such that communication regularity can be maximally used and the complexity of the architecture is minimized.

4.2. Optimization

4.2.1. Access Set Decomposition

Since for every pair of partitions $\mathbf{A}$ and $\mathbf{B}$ we have $(\mathbf{A} \odot \mathbf{B} \succeq \mathbf{A}) \wedge (\mathbf{A} \odot \mathbf{B} \succeq \mathbf{B})$ it makes sense to decompose the set of access functions in F into a set of subsets $G \subseteq \mathcal{P}F$ such that $F = \bigcup_S^G S$ and such that the original data access partition $\mathcal{Q}_F(\mathbf{C})$ is split into several data access partitions that are potentially finer. This may allow to utilize communication regularity for some of the subsets even if it cannot be utilized for $\mathcal{Q}_F(\mathbf{C})$. The access sets can only be decomposed in this way if consistency for the entire computation can be guaranteed.

In the context of Kahn Process Networks (KPNs), an array of data accessed by a pair of statements is considered as a communication channel and the access functions that produce and consume the data are mapped to input and output port domains of this channel [7,8,9,10,11]. Inspired by the producer-consumer view, we will minimize the number of access functions per subset by considering only those access functions within a basic producer-consumer set (PCS) and associating each basic PCS with a separate communication partition. The basic PCSs in G that result in a finer partitioning than the coarser partition in the hierarchy are then used to construct a new level of the access set topology. If two basic PCSs result from the same parent in the communication tree and yield the same communication partition, then they can be replaced by their union and the PCSs are not constructed separately.

For KPNs there is no global schedule and consistency of the computation is guaranteed by serializing the data that is communicated through a channel w.r.t. the original sequential specification of the computation using blocking read and write operations. Since a global schedule does exist in the spacetime hierarchy, we can model a communication channel as a simple memory without synchronisation. Since the dependencies are guaranteed to be satisfied for the global schedule the consistency of the computation is also guaranteed.

Evidently, subsets of access functions that access disjoint subsets of data elements can be considered separately. We can further minimize the size of a basic PCSs by creating one PCS $\hat{f}$ for every distinct read access function f. By including all write access functions in each PCS we can ensure that the correct data is read if the global spacetime schedule is obeyed. Let $\mathcal{V}$ denote an index set for $\Delta = \bigcup_{v}^{\mathcal{V}} \Delta_v$ such that a set $\mathcal{R}^v$ of access functions can be associated to each $v \in \mathcal{V}$ such that the access functions in $\mathcal{R}^v$ access data elements in Δ_v only. Let $\mathcal{W}^v \subseteq \mathcal{R}^v$ denote the subset of write access functions. For every access function f a basic PCS $\hat{f}$ is defined as

$$\forall_{v}^{\mathcal{V}} \; \forall_{f}^{\mathcal{R}^v} \; \hat{f} \triangleq \mathcal{W}_s^v \cup \{f\} \tag{9}$$

If no read accesses are available for a variable, the set of write accesses is considered as a separate PCS.

One may try to further decompose the PCSs by also separating the writing access functions into separate channels. To make this possible, all reading access functions must be altered to ensure that data is read from the right channel. While this may add complexity at run-time, in some cases the benefits of improved communication regularity may justify such an approach.

4.2.2. Constructing an Optimized Architecture

The first step in the construction of an optimized architecture for a given computation is to extract maximal parallelism by recursively constructing an alternating spacetime hierarchy $\phi \in \mathcal{H}_{\succeq} \Omega$ in the considered partition abstraction starting with $\phi_0 = \top_\Omega$. Each step $j+1$ of this recursion consists of two steps:

- Space partitioning: detect the finest partition $\phi_{2j+1} \preceq \phi_{2j}$ such that the subcells within every cell of ϕ_j access distinct cells of the data elements.

- Time partitioning: detect a coarsest partition $\phi_{2j+2} \prec \phi_{2j+1}$ such that within every cell of ϕ_j, the subcells can be totally ordered in a way that is compatible with $\rightsquigarrow$.

In the next step, each partition in the alternating spacetime hierarchy is coarsened to a partition of $\mathbf{\Omega}$ that maximizes communication regularity. For a given set G of PCSs obtained after access set decomposition at the level (g, h) we therefore first construct a set $H \subseteq G$ for which every PCS has a data access partition that is finer than the spacetime partition considered at that level:

$$H_{g,h} \triangleq \{K | K \in G \wedge \mathcal{Q}_K(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}}) \prec \mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}}\} \tag{10}$$

Since $\mathbf{A} \preceq \mathbf{B} \Rightarrow \mathbf{A} \odot \mathbf{B} = \mathbf{B}$, a useful strategy to utilize communication regularity is to find a set $M_{g,h} \subseteq H_{g,h}$ and a partition $\mathbf{\Omega}_{g+1}^x$ such that either

$$\bigvee_{N}^{M_{g,h}} \mathbf{\Omega}_{g+1}^x \otimes \mathbf{\Omega}_h^{\mathring{x}} \preceq \mathcal{Q}_N(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}}) \tag{11}$$

or

$$\bigvee_{N}^{M_{g,h}} \mathcal{Q}_N(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}}) \preceq \mathbf{\Omega}_{g+1}^x \otimes \mathbf{\Omega}_h^{\mathring{x}} \tag{12}$$

by choosing

$$\mathbf{\Omega}_{g+1}^x \preceq \bigotimes_{N}^{M_{g,h}} \mathcal{Q}_N(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}}) \tag{13}$$

or

$$\mathbf{\Omega}_{g+1}^x \succeq \bigodot_{N}^{M_{g,h}} \mathcal{Q}_N(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}}) \tag{14}$$

because communication regularity can then be used for every $N \in M_{g,h}$.

If L is the set of valid extensions of an alternating spacetime partition ϕ_k at a level of the hierarchy, then we can find $M_{g,h}$ by progressively adding new PCSs from $H_{g,h}$ while $\{\mathbf{C} | \mathbf{C} \in L \wedge \mathbf{C} \preceq \bigotimes_{N}^{M_{g,h}} \mathcal{Q}_N(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}})\} \neq \emptyset$ (or, $\{\mathbf{C} | \mathbf{C} \in L \wedge \mathbf{C} \succeq \bigodot_{N}^{M_{g,h}} \mathcal{Q}_N(\mathbf{\Omega}_g^x \otimes \mathbf{\Omega}_h^{\mathring{x}})\} \neq \emptyset$, respectively). The PCSs with the highest data-to-operations ratio in $H_{g,h}$ are considered first such that the optimization starts with the most complex communication partitions. Once a set $M_{g,h}$ has been constructed, the algorithm is restarted to find new solutions starting from the remaining PCSs in $H_{g,h}$. After completion the solutions are compared and the optimal solution is chosen.

Since communication regularity at a coarser level of the hierarchy also reduces the number of data cells for all finer levels of the hierarchy, an optimization that starts by optimizing the coarsest levels of h and progresses to finer levels appears to be an interesting strategy. When no more PCSs from $H_{g,h}$ can be added, the next h-level is considered to attempt to further improve the partition.

5. Architecture Instantiation

Instantiating the architecture requires to scan all elements in the partition hierarchy. For a partition hierarchy, the scanning proceeds hierarchically by scanning the coarsest partition and recursively scanning the partition of every cell it contains. Branches in partition trees are also scanned recursively.

To instantiate $\mathbf{\Omega}$, a specification of $\mathbf{\Omega}^{\mathfrak{s}}$ or code to scan $\mathbf{\Omega}^{\mathfrak{s}}$ is generated. If every processing element must be constructed separately the scanning code is executed to generate every processing element. Code to scan $\mathbf{\Omega}^{\mathfrak{t}}$ is also generated and provided as code to execute on every processing element (while an identification of the specific space cell is provided as parameters to the code).

To instantiate $\mathbf{\Delta}$, a specification of $\mathbf{\Delta}^{\mathfrak{s}}$ or code to scan $\mathbf{\Delta}^{\mathfrak{s}}$ is generated. If every interconnection element must be constructed separately the scanning code is executed and every cell of the interconnection hierarchy is instantiated while connecting the associated processing elements to it. Furthermore code must be added to copy data between the coarser and finer memory elements of $\mathbf{\Delta}^{\mathfrak{t}}$ between subsequent time steps.

6. Application example: CYK bifurcation

Many important algorithms that have been developed to analyze genomic data are closely related to the CYK CFG parsing algorithm. The most relevant part of this algorithm (for the purpose of optimization) can be specified in the polyhedral model. In this model, the sets of operations and data elements are represented by unions of $\mathcal{Z}$-polyhedra[2] where every operation is identified by its iteration vector. The data used by an operation is accessed through affine functions on the vector spaces that contain the sets of operations.

The CYK algorithm contains both uniform and non-uniform dependences. In contrast to the non-uniform dependences, the uniform dependences do not result in significant complexity for the communication architecture and little can be done to optimize for them. For this reason, we will restrict ourselves to the study of the non-uniform dependences. In order to enable a manual analysis of these dependences, we consider the simplest possible form of the CYK-algorithm that exhibits the same non-uniform communication patterns as the general CYK-algorithm. We therefore consider a trivial grammar that consists of the CFG-rule $S \rightarrow SS$ only, where S is a variable of the language. The corresponding loopnest is

$$\bigoTheta_{l=2}^{N} \bigoTheta_{i=0}^{N-l} \bigoTheta_{k=1}^{l-1} v[l,i] \triangleq v[l,i] + v[k,i] * v[l-k, i+k] \tag{15}$$

where Θ denotes a for loop.

The considered abstraction of partitions is the set of affine partitions of the sets of operations and data elements. The relevant operations on the partitions can be calculated using the linear algebra of vector spaces. Let us use $\mathcal{J}(f_1, f_2, \ldots, f_n)$ to succinctly denote the partition $\mathbb{A}_{[(f_1, f_2, \ldots, f_n)]}$ induced by a multi-dimensional affine function $(f_1, f_2, \ldots, f_n)$ on a set $\mathbb{A}$.

[2] a $\mathcal{Z}$-polyhedron is the intersection of a polyhedron with a grid

In section 6.1 the communication optimization algorithm is applied to the loopnest given above. Based on the observation that the available communication regularity is not optimally used, we use ad hoc transformations based on the algebraic properties of the computation in order to allow us to optimally use the available communication regularity in section 6.2. I believe that the resulting communication structure recovers a crucial part of a recent custom design of the Nussinov algorithm based on decades of research experience [12].

The similarity between the CYK parsing algorithm and other computationally more demanding bio-informatics algorithms (such as pairwise secondary structure alignment) suggests that the analysis presented in this paper might open the door to applying similar optimization steps to these algorithms.

6.1. Automatic Parallelisation and Communication Optimization

The given loopnest is the result of parallelisation using a synchronisation-minimal space-time partitioning [5,6]. The outer loop corresponds to a time partition that maximizes parallelism. The middle loop corresponds to a space partition, while the inner loop corresponds to another time partition. We will consider the given parallelism structure as an alternating spacetime partition that we will complete to obtain a spacetime hierarchy that optimizes communication.

The computation contains three distinct access functions:

$$\alpha(l,i,k) \triangleq [l,i] \quad \delta(l,i,k) \triangleq [k,i] \quad \epsilon(l,i,k) \triangleq [l-k, i+k] \tag{16}$$

We have three read references for a single statement s, which results in three PCSs $\hat{\alpha} = \{\alpha\}$, $\hat{\delta} = \{\alpha, \delta\}$ and $\hat{\epsilon} = \{\alpha, \epsilon\}$. Let a and b denote the first and second dimension of the data space respectively.

The given loopnest is the result of parallelisation using a classical spacetime partitioning technique. The outer loop corresponds to a time partition that maximizes parallelism. The middle loop corresponds to a space partition, while the inner loop corresponds to another time partition. We will consider the given parallelism structure as an alternating spacetime partition that we will complete to obtain a spacetime hierarchy that optimizes communication.

The data access partitions are:

$$\begin{aligned} \mathcal{Q}_{\hat{\alpha}}(\mathbf{\Omega}_1^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) &= \mathcal{J}(l,i) \\ \mathcal{Q}_{\hat{\delta}}(\mathbf{\Omega}_1^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) &= \mathcal{J}(i) \\ \mathcal{Q}_{\hat{\epsilon}}(\mathbf{\Omega}_1^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) &= \mathcal{J}(l+i) \end{aligned} \tag{17}$$

(Note that $\mathbf{\Omega}_1^{\mathfrak{t}} = \top_\Omega$ and $\mathbf{\Omega}_1^{\mathfrak{s}} = \top_\Omega$.) For the outer time partition, we choose the only available partition, $\mathbf{\Omega}_2^{\mathfrak{t}} \triangleq \mathcal{J}(l)$. This choice has communication regularity for $\hat{\alpha}$ only, since $\mathcal{K}^{\mathfrak{s}}_{1,1}(\hat{\alpha}) = \mathcal{J}(l) \prec \top_\Omega$ while $\mathcal{K}^{\mathfrak{s}}_{1,1}(\hat{\delta}) = \top_\Omega$ and $\mathcal{K}^{\mathfrak{s}}_{1,1}(\hat{\epsilon}) = \top_\Omega$. The data partition that corresponds to the memory partition for $\hat{\alpha}$ is $\mathcal{J}(a)$.

For the space partition, the available partitions that result in the same alternating spacetime partition are $L_0 = \{\mathcal{J}(i + \mu l) | \mu \in \mathbb{Q}\}$. The sets of PCSs that result from executing the optimisation algorithm are $M^{(0)}_{1,1} \triangleq \{\hat{\delta}\}$ and $M^{(1)}_{1,1} \triangleq \{\hat{\epsilon}\}$ where $M^{(0)}_{1,1}$ re-

sults in a partition $\mathbf{\Omega}_2^{\mathfrak{s}}(M_{1,1}^{(0)}) \triangleq \mathcal{Q}_{\hat{\delta}}(\mathbf{\Omega}_1^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) = \mathcal{J}(i)$ while $M_{1,1}^{(1)}$ results in a partition $\mathbf{\Omega}_2^{\mathfrak{s}}(M_{1,1}^{(1)}) \triangleq \mathcal{Q}_{\hat{\epsilon}}(\mathbf{\Omega}_1^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) = \mathcal{J}(l+i)$. Since these choices are completely symmetric and result in an architecture with the same complexity, we will only consider $M_{1,1}^{(0)}$. For the outer level of the time hierarchy, $\mathbf{\Omega}_2^{\mathfrak{s}}(M_{1,1}^{(0)})$ partitions both the $\hat{\alpha}$-channel with partition $\mathcal{J}(b)$ and the $\hat{\delta}$-channel with partition $\mathcal{J}(b)$. For the next level of the time hierarchy, $\mathbf{\Omega}_2^{\mathfrak{s}}(M_{1,1}^{(0)})$ further partitions the $\hat{\epsilon}$-channel with partition $\mathcal{J}(a+b)$.

Finally, for the inner time partition, the available partitions that result in the same alternating spacetime partition are $L_1 = \{\mathcal{J}(k+\gamma i) | \gamma \in \mathbb{Q}\}$. The $\hat{\alpha}$-channel is already partitioned completely. Since $\mathcal{Q}_{\hat{\delta}}(\mathbf{\Omega}_2^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) = \perp_\Omega$ and $\mathcal{Q}_{\hat{\epsilon}}(\mathbf{\Omega}_2^{\mathfrak{t}} \otimes \mathbf{\Omega}_1^{\mathfrak{s}}) = \perp_\Omega$ every possible partition has the same cost. For this reason we take the original schedule k.

Since the chosen spacetime hierarchy corresponds to the original layout of the loopnest, no changes must be made to its structure. The communication partitions can be used on a dynamic architecture that can benefit from locality automatically using local caches by converting the communication channels to separate variables and using the data layout suggested by the communication hierarchy. The original variable v is converted into three variables $v_\alpha, v_\delta, v_\epsilon$ and the data layout is transformed for every variable to reflect the correspond communication hierarchy. The chosen memory partition for v_α reduces it to a one-dimensional array. The resulting loopnest is

$$\bigoplus_{l=2}^{N} \bigoplus_{i=0}^{N-l} \bigoplus_{k=1}^{l-1} \begin{bmatrix} v_\alpha[i] \triangleq v_\alpha[i] + v_\delta[i,k] * v_\epsilon[l+i, l-k] \\ v_\delta[i,l] \triangleq v_\alpha[i] \\ v_\epsilon[l+i, l] \triangleq v_\alpha[i] \end{bmatrix} \tag{18}$$

Preliminary experiments on an 8-core UltraSparc T1 which can execute up to 32 threads simultaneously and on a dual-core Pentium D indicate the performance for this kernel is improved significantly for larger problem sizes and is more scalable compared to classical approaches.

If we want to implement the resulting loopnest on an architecture such as an FPGA where the memories and interconnections are managed explicitly, then the space partition allows to reduce the $\hat{\alpha}$-channel to a set of scalars and the $\hat{\delta}$-channel to a set one-dimensional arrays that must be accessible to single processsing elements only. However, since the $\hat{\epsilon}$-channel is only partitioned at the second level of the time hierarchy, the data in the $\hat{\epsilon}$-channel can only be moved to one-dimensional arrays within each iteration of l and must be copied between the one-dimensional arrays and a two-dimensional array that is accessible by each of these channels before and after the execution of the operations within an l-time cell. The cost of this solution might drastically reduce the performance benefit obtained through parallelisation. At the same time, the analysis of the data access partitions in equation (17) indicates both the $\hat{\delta}$-channel and $\hat{\epsilon}$-channel have communication regularity at the coarsest level of the computation. It is evident to ask whether it is possible to transform the original loopnest with ad hoc transformations to make this communication regularity usable at the coarsest level.

6.2. Improved Optimization Enabled by Ad Hoc Transformations

Closer inspection of the dependencies that constrain the set of solutions for the coarsest time partition reveals that every inner k-loop for l always depends on the last iteration of

a k-loop for $l-1$ through one of the main communication channels. Since the algebraic properties of the reduction performed in the inner loop allow to reorder the operations in this loop, we use an ad hoc index set splitting transformation that splits the iteration space in two parts,

$$\underset{l=2}{\overset{N}{\Theta}}\ \underset{i=0}{\overset{N-l}{\Theta}} \left[\begin{array}{l} \underset{k=1}{\overset{2k\leq l}{\Theta}} v[l,i] \triangleq v[l,i] + v[k,i] * v[l-k,i+k] \\ \\ \underset{2k>l}{\overset{k\leq l-1}{\Theta}} v[l,i] \triangleq v[l,i] + v[k,i] * v[l-k,i+k] \end{array} \right. \tag{19}$$

and time-reverse the k-loop of the part where the k-loop starts with the value computed at the previous l-value:

$$\underset{l=2}{\overset{N}{\Theta}}\ \underset{i=0}{\overset{N-l}{\Theta}} \left[\begin{array}{l} \underset{2k\leq l}{\overset{k\geq 1}{\Theta}} v[l,i] \triangleq v[l,i] + v[k,i] * v[l-k,i+k] \\ \\ \underset{2k>l}{\overset{k\leq l-1}{\Theta}} v[l,i] \triangleq v[l,i] + v[k,i] * v[l-k,i+k] \end{array} \right. \tag{20}$$

In this new loopnest the first iteration of the k-loops at length $l = l_a$ require data computed at lengths $l = \left\lfloor \frac{l_a}{2} \right\rfloor$ or $\left\lfloor \frac{l_a+1}{2} \right\rfloor$ and only the last iteration requires data computed at lengths $l = 1$ and $l = l_a - 1$. The iteration at $l = l_a$ can therefor start long before the iteration at $l_a - 1$ completes. The two statements significantly increase the complexity for parallelisation analysis. In order to make it amenable to manual analysis, we will fuse the two inner loops into a single loop by combining operations that depend on the same lengths l. We first substitute the iterator of the second loop with $q = l - k$,

$$\underset{l=2}{\overset{N}{\Theta}}\ \underset{i=0}{\overset{N-l}{\Theta}} \left[\begin{array}{l} \underset{2k\leq l}{\overset{k\geq 1}{\Theta}} v[l,i] \triangleq v[l,i] + v[k,i] * v[l-k,i+k] \\ \\ \underset{2q<l}{\overset{q\geq 1}{\Theta}} v[l,i] \triangleq v[l,i] + v[l-q,i] * v[q,i+l-q] \end{array} \right. \tag{21}$$

and subsequently fuse the inner loops and the statements contained by them into one loop and one statement using $g = k = q$,

$$\underset{l=2}{\overset{N}{\Theta}}\ \underset{i=0}{\overset{N-l}{\Theta}}\ \underset{2g\leq l}{\overset{g\geq 1}{\Theta}} \left[\begin{array}{l} v[l,i] \triangleq v[l,i] + v[g,i] * v[l-g,i+g] \\ \qquad\qquad + (2g < l)?(v[l-g,i] * v[g,i+l-g]) : 0 \end{array} \right. \tag{22}$$

I believe this new loopnests permits two dimensions of parallelism after a first time-partition and that these two dimensions of parallelism will allow to use all communicaton regularity at the coarsest level so that the resulting architecture is ideally suited for an FPGA implementation.

7. Conclusion

This paper is a slightly modified version of a draft paper that was submitted to ParCo 2011 and is very preliminary. Since I do not have the resources to complete this paper

by increasing its clarity, extending the experimental evaluation and adding a section on related work, I'm making it available so that it may be useful to others.

8. Acknowledgements

I am grateful to Sean Rul for helping me setup the Niagara experiments.

Initial research that provided the starting point for this paper was supported in part by a PhD grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT-Vlaanderen)[3] and a BOF/GOA project[4] and was also morally supported by the Flexware (IWT/060068) project.

References

[1] Anderson, J.M., Amarasinghe, S.P., Lam, M.S.: Data and computation transformations for multiprocessors. In: Proc. 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'95, Santa Barbara, California (1995) 166–178

[2] Anderson, J.: Automatic Computation and Data Decomposition for Multiprocessors. PhD thesis, Stanford, CA, USA (1997)

[3] Lim, A., Liao, S., Lam, M.: Blocking and array contraction across arbitrarily nested loops using affine partitioning. Proceedings of the eighth ACM SIGPLAN symposium on Principles and practices of parallel programming (2001) 103–112

[4] Liao, S.: SUIF Explorer: An Interactive and Interprocedural Parallelizer. PhD thesis, Stanford University (2000)

[5] Lim, A.W., Lam, M.S.: Maximizing parallelism and minimizing synchronization with affine transforms. In: Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM Press (1997) 201–214

[6] Lim, A.W., Lam, M.S.: Maximizing parallelism and minimizing synchronization with affine partitions. Parallel Computing **24**(3–4) (May 1998) 445–475

[7] Kienhuis, B.: Compaan: Deriving process networks from matlab for embedded signal processing architectures. In: In Proceedings of the 8th International Workshop on Hardware/Software Codesign (CODES. (2000) 13–17

[8] Rijpkema, E., Kienhuis, B., Deprettere, E.: Compilation from Matlab to process networks. Second International Workshop on Compiler and Architecture Support for Embedded Systems (CASES'99) (1999)

[9] Turjan, A., Kienhuis, B., Deprettere, E.: Solving out-of-order communication in kahn process networks. The Journal of VLSI Signal Processing **40** (2005) 7–18

[10] Turjan, A., Kienhuis, B., Deprettere, E.: Translating affine nested-loop programs to process networks. In: CASES '04: Proceedings of the 2004 international conference on Compilers, architecture, and synthesis for embedded systems, New York, NY, USA, ACM Press (2004) 220–229

[11] Turjan, A., Kienhuis, B., Deprettere, E.: Classifying interprocess communication in process network representation of nested-loop programs. Trans. on Embedded Computing Sys. **6**(2) (2007) 13

[12] Jacob, A., Buhler, J., Chamberlain, R.D.: Accelerating nussinov rna secondary structure prediction with systolic arrays on fpgas. In: Proceedings of the 2008 International Conference on Application-Specific Systems, Architectures and Processors, Washington, DC, USA, IEEE Computer Society (2008) 191–196

[3] from 01/01/2008 to 31/08/2009

[4] from 01/07/2006 to 31/12/2007