

PCA Machine Model

Version 1.0

July 19, 2004



©2004 Georgia Tech Research Corporation, all rights reserved.

A non-exclusive, non-royalty bearing license is hereby granted to all persons to copy, modify, distribute and produce derivative works for any purpose, provided that this copyright notice and following disclaimer appear on all copies: THIS LICENSE INCLUDES NO WARRANTIES, EXPRESSED OR IMPLIED, WHETHER ORAL OR WRITTEN, WITH RESPECT TO THE SOFTWARE OR OTHER MATERIAL INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE, OR ARISING FROM A COURSE OF PERFORMANCE OR DEALING, OR FROM USAGE OR TRADE, OR OF NON-INFRINGEMENT OF ANY PATENTS OF THIRD PARTIES. THE INFORMATION IN THIS DOCUMENT SHOULD NOT BE CONSTRUED AS A COMMITMENT OF DEVELOPMENT BY ANY OF THE ABOVE PARTIES.

This material is based in part upon work supported by the U.S. Defense Advanced Research Projects Agency (DARPA) and other agencies of the U.S. Department of Defense (DoD). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or the DoD.

The US Government has a license under these copyrights, and this material may be reproduced by or for the U.S. Government.



Acknowledgements

Authors

The primary author of this document was:

Peter Mattson Reservoir Labs, Inc.

The author wishes to thank the members of the Morphware Forum who reviewed and contributed to this document. The author also thanks the Defense Advanced Research Projects Agency (DARPA) for their support of this work.

The Morphware Forum

The Morphware Forum is a joint activity of the participants in DARPA's Polymorphous Computing Architectures (PCA) program, as well as other interested developers of embedded computing hardware, software, and application technology. The purpose of the Morphware Forum is to define an open, portable software environment for the development of high performance applications on PCA platforms. Morphware Forum products and information are available at www.morphware.org.

The following organizations are voting members of the Morphware Forum at this writing:

- Defense Advanced Research Projects Agency
- Georgia Institute of Technology
- Lockheed Martin Advanced Technology Laboratory
- Mercury Computing
- Massachusetts Institute of Technology
- MIT Lincoln Laboratory
- MPI Software Technology, Inc.
- Protean Devices, Inc.
- Reservoir Labs, Inc.
- Stanford University
- University of Texas, Austin
- University of Southern California Information Sciences Institute

Additional contributing organizations include:

- Air Force Research Laboratory
- Applied Photonics
- BAE
- Brigham Young University
- California Institute of Technology
- George Mason University
- IBM Austin Research Laboratory
- IBM T. J. Watson Research Center
- Lockheed Martin Aerospace
- Lockheed Martin NE&SS
- Los Alamos National Laboratory
- Mississippi State University
- North Carolina State University
- Northrop Grumman
- Raytheon
- Sandia National Laboratory
- South West Research Institute
- Space and Naval Warfare Systems Center San Diego
- University of California, Berkeley
- University of California, Irvine
- University of Illinois at Urbana-Champaign
- University of Maryland
- University of Pennsylvania
- Vanderbilt University
- Vermont University
- VLSI Photonics

Document Change History

Version 1.0 is the first approved version of the document.

Table Of Contents

Acknowledgements.....	i
Authors.....	i
The Morphware Forum.....	i
Document Change History.....	iii
Table Of Contents.....	iv
1. Overview.....	1
2. Objects	3
2.1. Root Object.....	3
2.2. Primary Objects	3
2.3. Secondary Objects	3
2.4. Tertiary Objects	4
2.5. Reference Objects.....	4
3. Machine Model Parameters.....	5
3.1. Common Object Parameters	5
3.2. Root Parameters	5
3.3. Ingredient Parameters	6
3.4. Processor Parameters.....	6
3.5. Memory Parameters	10
3.6. Network Link Parameters	11
3.7. Morph Parameters.....	12
3.8. Reference Parameters.....	15
4. Arrays of Objects	16
4.1. References to Objects in Arrays	16
4.2. Abbreviated Reference to Objects in Arrays.....	17
4.3. Array Example	17
5. Topology.....	19
5.1. Definitions.....	19
5.2. Processor Topology.....	19
5.3. Cache Topology	20
6. References	21

Appendix: Machine Model XML Schema22

1. Overview

The Polymorphous Computing Architectures (PCA) program is developing new hardware and software for high performance, high-flexibility embedded computing. The portable application development environment for PCA architectures is called the Morphware Stable Interface (MSI). An introduction to the PCA program and the MSI is given in [1].

The PCA Machine Model is a generic model used to describe a specific PCA architecture. The phrase “the machine model” refers to the model. Phrases like “a machine model” or “the TRIPS machine model” refer to a description of a specific architecture using the machine model. The primary role of the machine model is to describe the architecture for software tools common to all PCA architectures, such as a component manager or high-level compiler.

The goal of the machine model is to describe the architecture in a simple, uniform manner that clearly defines functionality and enables accurate performance estimates. Toward this end, it describes capabilities rather than implementation and makes quantitative rather than qualitative distinctions whenever possible.

A machine model contains three key kinds of components:

1. A *resource* is a discrete device with well-defined functionality. There are three kinds of resources in the machine model: processors, memories, and network links.
2. An *ingredient* is an arbitrary, configurable division of underlying hardware. An ingredient can be configured, possibly in concert with other ingredients, into one or more resources.
3. A *morph* defines a specific set of ingredients that can be configured into a specific set of resources.

For example, a hypothetical “PCAPower” machine model divides the underlying hardware into three ingredients:

- “Foo” ingredient
- “Bar” ingredient
- “Baz” ingredient

The ingredients can be configured into several resources:

- “P1000” processor
- “P2000” processor
- “P3000” processor
- “MX” memory
- “NX” network link

Four morphs describe how the ingredients can be configured into resources:

- “IrregularComp” morph, configures the “Foo” ingredient into a “P1000” processor.

- “RegularComp” morph, configures the “Bar” and “Baz” ingredients into a “P2000” processor.
- “MaximumPower” morph, configures the “Foo”, “Bar”, and “Baz” ingredients into the “P3000” processor.
- “DefaultResources” morph, configures no ingredients into an “MX” memory and an “NX” network link. A “DefaultResources” morph is always available.

It may not be possible to configure the system to make all of the resources available at the same time. The available ingredients and morphs define the possible combinations of resources.

For example, in the “PCAPower” system, the “RegularComp” and IrregularComp” morphs can both be configured at the same time since they require disjoint ingredients. However, neither can be configured in conjunction with the “MaximumPower” morph since that morph demands all three ingredients.

The machine model is a conceptual model. It is part of the PCA metadata system [2] and is documented as an XML schema [3] and included in the appendix to this document.

2. Objects

The machine model is composed of a hierarchy of *objects*. An object is a structure that describes a part of the machine model with a collection of *parameters*. Each parameter may be a simple value (numeric, enumerated, or string), an object, or a list of values or objects.

The hierarchy of objects consists of one root object and multiple primary, secondary, tertiary, and reference objects. The *root object* represents the entire machine model. *Primary objects* represent the key components identified in the introduction: ingredients, resources, and morphs. Some parameters of the root object are primary objects. Some parameters of a primary object such as a processor may be *secondary objects*, such as functional units. Some parameters of secondary objects may be *tertiary objects*, such as pipeline stages. At all levels of the hierarchy, some parameters of an object may be *reference objects* that refer to objects elsewhere in the hierarchy.

2.1. Root Object

Object	Description
mm_Root	Single root object that represents the entire machine model.

2.2. Primary Objects

Object	Description
mm_Ingredient	An architecture-specific hardware ingredient.
mm_Proc	A potential processor resource.
mm_Mem	A potential memory bank resource.
mm_Net	A potential network link resource.
mm_Morph	Defines a set of hardware ingredients that can be configured to produce a set of resources.

2.3. Secondary Objects

Object	Description
mm_FU	Functional unit within a processor resource.
mm_Kernel	Predefined or black-box kernel supported by a processor resource.
mm_LatencyFromMorph	Specifies minimum latency to configure a morph if one or more ingredients were previously part of another specific morph.

2.4. Tertiary Objects

Object	Description
mm_Pipeline<category>	Lists occupied stages for a cycle of the operation <category>'s latency (e.g., PipelineIAdd, see Section 3.4.2).
mm_StreamInfo	Information related to the streams used in predefined and black-box kernels.
mm_BlockInfo	Information related to blocks used in predefined and black-box kernels.

2.5. Reference Objects

Object	Description
mm_Ref	Reference to another object.

3. Machine Model Parameters

3.1. Common Object Parameters

Certain parameters are common to multiple objects. These parameters appear below:

Parameter	Type – Description
All objects except Ref	
Name	string – Globally unique identifier of an object, composed only of alphanumeric characters and underscores.
All primary objects and FU	
Dimension	string – See Section 4.

3.2. Root Parameters

A single root object has parameters that contain the primary objects of the architecture, such as a list of processors, or apply to the system as a whole, such as the data layout.

Parameter	Type – Description
AddressSize	int32 – Size of addresses in bits.
WordSize	int32 – Size of words in bits.
BigEndian	boolean – True if data is Big Endian.
Alignment<type>	int32 – Alignments for char, short, int, long, float, double in bits (e.g., AlignmentChar, AlignmentShort).
AlignmentBitfield	int32 – Alignment of bitfields in bits (1 for packed bitfields).
BitfieldCrossWord	boolean – True if bitfields can cross word boundaries.
Ingredient	List of mm_Ingredient – All ingredients that compose the architecture.
Proc	List of mm_Proc – All processors the ingredients could be configured into.
Mem	List of mm_Mem – All memories the ingredients could be configured into.
Net	List of mm_Net – All network links the ingredients could be configured into.
Morph	List of mm_Morph – Definitions of all the possible morphs of the architecture.

3.3. Ingredient Parameters

Ingredients are arbitrary portions of hardware that can be configured into resources. The only parameters of an ingredient are the common object parameters described in Section 3.1. The resources into which an ingredient can be morphed are described by the morphs.

3.4. Processor Parameters

A processor represents anything that consumes, produces, or moves data. For instance, a processor object can represent a Raw [4] tile, a DMA engine, or an I/O device.

Parameter	Type – Description
Master	List of mm_Ref to mm_Proc – If empty, the processor is a master processor. If not empty, the processor is a slave processor that executes kernels as requested by the master processors in the list. A slave processor may not appear in the master list of another processor.
SupportsUserCode	boolean – True if the processor can execute user code (including user-defined kernels).
SupportedKernel	List of mm_Kernel – Predefined and black-box kernels the processor can execute with performance metadata for each kernel.
ContextSwitch	int32 – Number of processor cycles lost to a context switch (master processor) or kernel switch (slave processor).
Multiplexing	int32 – Maximum number of predefined kernels that can be multiplexed on the processor.

3.4.1. User-code processor parameters

The following parameters are used only for processors running user code.

Parameter	Type – Description
Freq	float – Frequency of processor in cycles/nanosecond. For processors that support a configurable processor frequency, this parameter provides a list of valid processor frequency settings.
Supports<type>	boolean – True if processor supports operations on char, short, int, long, float, double (e.g., SupportsChar, SupportsShort).
FU	List of mm_FU – All functional units contained within this processor.

Parameter	Type – Description
BranchMispredict	int32 – Number of processor cycles lost to a branch misprediction.
NumRegs	int32 – Number of registers in the processor.
ISize	int32 – Number of bits required to store an operation for this processor.
NumSIMDClusters	int32 – Number of parallel SIMD execution engines. Speed up of data parallel code equal to number of engines. Also used for number of clusters when determining if access is inlane per Mem.RAMInlaneBw defined in Section 3.5.
SIMDSubword	boolean – True if processor can execute a number of operations equal to (largest type width / actual type width) when dealing with smaller data widths.
LLCQuality	float – Ratio of estimated schedule length to use for this architecture to estimated schedule length per performance model. Adjustment factor for low-level compiler scheduling or unmodeled hardware constraints.
LLCSwPipelineQuality	float – Ratio of estimated schedule length to use for this architecture to estimated schedule length per performance model. Adjustment factor for quality of low-level compiler software pipelined scheduling or unmodeled hardware constraints. Zero indicates that the low-level compiler does not software pipeline.

3.4.2. Functional Unit Parameters

A processor that supports user code must contain one or more functional units. Each functional unit is capable of a certain set of C operations. For a given functional unit, each operation category has a certain latency and a certain occupancy of pipeline stages within the functional unit.

Parameter	Type – Description
For all operation categories	
Latency<category>	int32 – Number of cycles the operation takes to complete, where a latency of zero indicates inability to perform the operation (e.g., LatencyIAdd – all possible <category> values are listed later in this section).
Pipeline<category>	List of mm_Pipeline<category> – Defines which pipeline stages are occupied for each cycle of latency, in

Parameter	Type – Description
	order (e.g., PipelineIAdd).

Functional units also have the following parameter.

Parameter	Type – Description
AccessesMem	List of mm_Ref to mm_Mem – Memories that this functional unit can access.

3.4.2.1. OPERATION CATEGORIES

This describes the operation categories used to define the above functional unit parameters.

Parameter	Type – Description
Bitwise	~, &, , ^, <<, >>, simple assignment
Select	Non-branching ?: Latency of zero indicates pervasive predication support
IComp	<, <=, >=, >, ==, !=
IAdd	+, -
IMul	*
IDiv	/, %
FPComp	Floating point <, <=, >=, >, ==, !=
FPAdd	Floating point +, -
FPMul	Floating point *
FPDiv	For floating point /, %
Load	Load operations from RAM
Store	Store operations to RAM
Peek	Hardware implemented peek
Pop	Hardware implemented pop
Push	Hardware implemented push

3.4.2.2. PIPELINE<CATEGORY> PARAMETERS

A Pipeline<category> object lists the pipeline stage numbers occupied by an operation in that category on a given cycle.

Parameter	Type – Description
OccupiedStage	List of int 32 – Stage numbers that are occupied for this cycle.

The first cycle must occupy at least stage zero, and the last cycle must occupy at least stage one. All other stages may be used arbitrarily.

3.4.3. Kernel Parameters

Predefined and black-box kernels are modeled as possible software-pipelined loops, with a certain duration for the prologue before the loop and for the epilogue after the loop. Iterations of the loop have an initiation interval and latency. Each iteration reads a certain number of bytes from input streams and writes a certain number of bytes to output streams. Predefined and black-box kernels specify the memories in which their streams and blocks reside, but not the specific address of the blocks or streams (which is determined by the high-level compiler).

Parameter	Type – Description
Multiplex	boolean – True if kernel can be multiplexed on a processor.
PrologueTime	float – Run-time before start of first iteration, or total run time if kernel does not use streams (in nanoseconds).
EpilogueTime	float – Run-time after iteration, or zero if kernel does not use streams (in nanoseconds).
InitiationInterval	float – Delay between the start of successive iterations (in nanoseconds).
Latency	float – Length of one iteration (in nanoseconds).
StreamInfo	List of mm_StreamInfo – Information on streams read/written by kernel. See section 3.4.3.1.
BlockInfo	List of mm_BlockInfo – Information on blocks read/written by kernel. See section 3.4.3.2.

3.4.3.1. STREAMINFO

Information about a stream that is read or written by a predefined or black-box kernel.

Parameter	Type – Description
StoredInMem	mm_Ref to mm_Mem – Reference to the memory where the stream is stored.
Size	int32 – Size of the stream (in bytes) in RAM, zero for streams in FIFOs.
InputRate	int32 – Average number of bytes read per “iteration.”
OutputRate	int32 – Average number of bytes written per “iteration.”

3.4.3.2. BLOCKINFO

Consists of the following parameters.

Parameter	Type – Description
StoredInMem	mm_Ref to mm_Mem – Reference to the memory where the block is stored.
Size	int32 – Size of the block (in bytes).
InputRate	int32 – Average number of bytes read per “iteration.”
OutputRate	int32 – Average number of bytes written per “iteration.”

3.5. Memory Parameters

A memory represents anything that stores data.

Parameter	Type – Units Interpretation
Subtype	{ FIFO, RAM, CACHE } – Mode of operation supported by this memory resource.
Size	int64 – Size of the memory (number of bytes available to store instructions and/or data).

3.5.1. FIFO parameters

Parameter	Type – Units Interpretation
FIFOpeek	int32 – The number of bytes in FIFO that can be peeked.

3.5.2. RAM parameters

Parameter	Type – Units Interpretation
RAMCoherence	{ COHERENT, INCOHERENT, UNCACHEABLE } – Specifies level of support offered by caches connected to this RAM.
RAMStreamEfficientSize	int32 – Minimum size of a stream in bytes for which DMA flow between simultaneous producer and consumer is efficient (achieves at least 95% of stated bandwidth).
RAMLinearBw	float – Bandwidth for doing linear accesses into memory (normal stream pop/push) (bytes/nanosecond).
RAMRandomBw	float – Bandwidth for random access into memory (e.g., unpredictable peek, block read/write, etc.) (bytes/nanosecond).
RAMInlaneBw	float – Bandwidth for random inlane access into memory for clustered architectures (each cluster indexes in its own memory lane, i.e., address % num clusters == cluster index) (bytes/nanosecond).

3.5.3. Cache parameters

Parameter	Type – Units Interpretation
CacheContent	{DATA, INSTRUCTIONS, UNIFIED} – Specifies allowable contents of cache.
CacheCoherent	boolean – Indicates if the cache may be connected to a memory that must be coherently cached. No cache may be connected to a memory that is not cacheable.
CacheLinesize	List of int 32 – The size(s) of cache lines supported (bytes).
CacheAssociativity	int 32 – The associativity of the cache, i.e., the number of locations in the cache where a main memory word can be stored.
CacheWriteback	boolean – True if the cache is writeback, false if it is writethrough.

3.6. Network Link Parameters

Network links connect an arbitrary set of senders to an arbitrary set of receivers, and model the bandwidth and latency available along that connection.

One-way connections are described with disjoint sets of senders and receivers; two-way connections are described with the same set as senders and receivers. The sets of senders and receivers may contain one or multiple resources. Hence, a network link may represent a simple one-to-one connection or a bus.

Multiple network links can be arbitrarily connected to represent more complex structures and constraints on latency and bandwidth. For instance, a crossbar could be represented as one network link for each of N inputs and one network link for each of N outputs, all with bandwidth X , all connected through a common network link with bandwidth $N*X$. Such a cross bar is shown below:

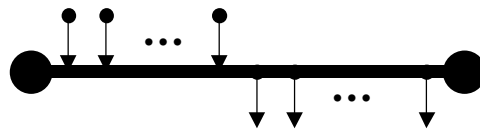


Figure 1. An example of a crossbar switch represented by multiple network links.

Connecting one network link to another is assumed to be a reciprocal relationship. If both A and B are network links and B appears in A 's Sender list, then A is treated as appearing in B 's Receiver list even if it does not. Similarly, if B appears in A 's Receiver list, then A is treated as appearing in B 's Sender list even if it does not. This allows network links to be connected to the edges of arrays (see Section 4) of network links, for instance.

Network links may connect resources that exist in different morphs. References to resources that are not among the current active morphs in the senders and receivers sets are ignored. For example, suppose a processor tile ingredient can morph into six different processors and a memory tile ingredient can morph into three different kinds of memories, but any processor can be connected to any memory. A single network link that includes the six potential processors and three potential memories in its Sender and Receiver lists can be used to describe the connection.

Parameter	Type – Description
Sender	List of mm_Ref to resource – All resources that can send data to this link.
Receiver	List of mm_Ref to resource – All resources that can receive data from this link.
Latency	float – Latency of this network hop under typical conditions. In the future, the machine model may also require minimum/maximum latencies (nanoseconds).
Bw	float – Bandwidth of the network link, measured as the maximum rate that data can be read or written over the link (bytes/nanosecond).

3.7. Morph Parameters

A morph expresses a set of ingredients that can be configured into a set of resources. One morph that uses no ingredients is allowed in any architecture, and it defines the resources that are always present. If two morphs produce the same resource, they must use at least one ingredient in common, which prevents both from being active at the same time.

Switching between morphs that use one or more of the same ingredients requires a certain latency, which depends on the morphs that previously used the ingredient(s). During this time, the resources contained in either morph must be unused. The state of resources that exist in one morph but not the other is lost; the state of resources that exist in both morphs is retained.

This definition creates a distinction between morphs and parameter settings: changing morphs (i.e., from a threaded to a streaming processor) destroys state, while changing settings (i.e., changing a processor's frequency from 600 MHz to 700 MHz) does not.

Parameter	Type – Description
ConfiguresIngredient	List of mm_Ref to mm_Ingredient – Ingredients used by the morph.
IntoResource	List of mm_Ref to resource – Resources produced by the morph.

Parameter	Type – Description
LatencyFromMorph	List of mm_LatencyFromMorph – Defines latency of morph based on the morphs that the ingredients previously belonged to.

3.7.1. LatencyFromMorph

Parameter	Type – Description
FromMorph	mm_Ref to mm_Morph – Morph that previously contained the ingredients used by this morph.
Latency	int32 – Minimum latency required to configure this morph if any of the ingredients were previously used by FromMorph (nanoseconds).

3.7.2. Morph Example

Consider a simple architecture consisting of two processor tile ingredients, *X1* and *X2*. *X1* can be configured into processor *PA1* or processor *PB1*. Likewise, *X2* can be configured into processor *PA2* or *PB2*. Both tiles may be configured together into processor *PC*. The architecture always contains a network link *N* and a memory *M*. The following example shows just the morphs (in pseudo-XML syntax). (Note: the Ingredient, Proc, Mem, and Net objects are omitted here, but are included in the example in section 4.3.)

```

Morph
  Name = "Y0"
  ConfiguresIngredient = {}
  IntoResource = { M, N }
Morph
  Name = "YA1"
  ConfiguresIngredient = { X1 }
  IntoResource = { PA1 }
  LatencyFromMorph
    FromMorph = YB1
    Latency = 15
  LatencyFromMorph
    FromMorph = YC
    Latency = 50
Morph
  Name = "YB1"
  ConfiguresIngredient = { X1 }
  IntoResource = { PB1 }
  LatencyFromMorph
    FromMorph = YA1
    Latency = 10
  LatencyFromMorph
    FromMorph = YC
    Latency = 50
Morph
  Name = "YA2"

```

MACHINE MODEL PARAMETERS

```

ConfiguresIngredient = { X2 }
IntoResource = { PA2 }
LatencyFromMorph
  FromMorph = YB2
  Latency = 15
LatencyFromMorph
  FromMorph = YC
  Latency = 50
Morph
  Name = "YB2"
  ConfiguresIngredient = { X2 }
  IntoResource = { PB2 }
  LatencyFromMorph
    FromMorph = YA2
    Latency = 10
  LatencyFromMorph
    FromMorph = YC
    Latency = 50
Morph
  Name = "YC"
  ConfiguresIngredient = { X1, X2 }
  IntoResource = { PC }
  LatencyFromMorph
    FromMorph = YA1
    Latency = 70
  LatencyFromMorph
    FromMorph = YB1
    Latency = 75
  LatencyFromMorph
    FromMorph = YA2
    Latency = 70
  LatencyFromMorph
    FromMorph = YB2
    Latency = 75

```

One can summarize the morphing information in a table:

Morph	Ingredients		Resources							Latencies (to morph from)					
	X1	X2	PA1	PB1	PA2	PB2	PC	M	N	Y0	YA1	YB1	YA2	YB2	YC
Y0	-	-	-	-	-	-	-	1	1	-	-	-	-	-	-
YA1	1	-	1	-	-	-	-	-	-	-	-	15	-	-	50
YB1	1	-	-	1	-	-	-	-	-	-	10	-	-	-	50
YA2	-	1	-	-	1	-	-	-	-	-	-	-	-	15	50
YB2	-	1	-	-	-	1	-	-	-	-	-	-	10	-	50
YC	1	1	-	-	-	-	1	-	-	-	70	75	70	75	-

This table makes relationships described by the morphs more obvious. Morphs *YA1* and *YB1* cannot both be active at the same time, since both require ingredient *X1*. If morphs *YA1* and *YB2* are active, and the high-level compiler decides to reconfigure into morph

YC, it estimates that the morph will take 75 nanoseconds (the maximum of 70 to morph *X1* from *YAI* and 75 to morph *X2* from *YBI*).

3.8. Reference Parameters

Every object in the machine model has a unique name. A reference object refers to another object by this name.

Parameter	Type – Units Interpretation
Name	string – Name of object that is referenced.
Index	string – See Section 4.

4. Arrays of Objects

All primary objects and FU objects within the machine model can represent arrays of identical objects. Such objects have a `Dimension` parameter. The `Dimension` parameter is empty for single objects, but can contain a series of dimensions separated by spaces to indicate an array of objects. For example, if an `mm_Proc` object had a `Dimension` parameter of “4 4”, it would indicate that the object is shorthand for a 4-by-4 array of identical objects. Arrays provide conciseness, not fundamental semantics – an array is equivalent to multiple individual objects.

4.1. References to Objects in Arrays

An `mm_Ref` object that refers to an array also needs to use the `Index` parameter to specify the indices of the specific object. The `Index` parameter is empty when referring to single objects, but must contain a series of indices separated by spaces when referring to an object in an array. One index is required for each dimension in the array. Array elements are indexed starting from zero.

The machine model supports three kinds of indices: *absolute*, *relative*, and *anonymous*.

1. **Absolute indices** are “normal” array indices. Absolute indices are specified as unsigned integers.
2. **Relative indices** are used by objects in one array to refer to corresponding objects in another array. If an object in an array of objects *A* refers to an object in another array of objects *B* using a relative index, the index of the object in *A* is added to the relative index to obtain the index of the object in *B* that is referred to. Relative indices are specified as unsigned integers preceded by a plus or minus sign enclosed in square brackets, where a plus sign specifies a non-negative relative index and a minus sign specifies a negative relative index.
3. **Anonymous indices** are used to refer to any instance of an ingredient when describing the ingredients used by a morph, or any instance of a morph when describing morph latencies, if all instances are equivalent. Anonymous indices are specified with an asterisk.

It is important to note that indices, like arrays, do not add fundamental semantics to the machine model. An equivalent machine model may be constructed without arrays or indices.

4.1.1. Examples of References to Objects in Arrays

The differences between the three kinds of indices are best shown by a few simple examples. For instance, given *A* in `Array1` and *B* in `Array2`, *A* could refer to *B* as:

Name = “B”, Index = “0”	instance I of A refers to instance [0] of B
Name = “B”, Index = “+0”	instance I of A refers instance [I] of B
Name = “B”, Index = “+1”	instance I of A refers instance [I+1] of B.
Name = “B”, Index = “*”	instance I of A refers to any instance [<i>any index</i>] of B.

Different kinds of indices can be combined to refer to objects in multi-dimensional arrays. For instance, if *A* and *C* are in two-dimensional arrays, *A* could refer to *C* as:

Name = "C", Index= "0 +3"	instance [<i>I_x</i> , <i>I_y</i>] of <i>A</i> refers to instance [0, <i>I_y</i> +3] of <i>C</i>
Name = "C", Index= "0 *"	instance [<i>I_x</i> , <i>I_y</i>] of <i>A</i> refers to instance [0,< <i>any index</i> >] of <i>C</i>

4.2. Abbreviated Reference to Objects in Arrays

Two conventions are supported for conciseness. A short ellipsis ".." may be used to refer to a range of absolute or relative indices. An asterisk may be followed by an unsigned integer *N* to indicate *N* anonymous references. The following simple examples demonstrate these conventions:

4.2.1. Examples of Abbreviated Reference to Objects in Arrays

Name = "B", Index = "4..6"	short for: Name = "B", Index = "4" Name = "B", Index = "5" Name = "B", Index = "6"
Name = "B", Index = "+0..+1"	short for: Name = "B", Index = "+0" Name = "B", Index = "+1"
Name = "B", Index = "*2"	short for: Name = "B", Index = "*" Name = "B", Index = "*"

4.3. Array Example

Returning to the example in section 3.7.2, the morphs can be described much more concisely using arrays (again, in pseudo-XML with references into arrays using the *Name[Index]* shorthand). (Note: This example also includes the `Proc`, `Mem`, and `Net` descriptions that were absent in section 3.7.2.)

```

Root
Ingredient
  Name = "X"
  Dimension = "2"
...
Proc
  Name = "PA"
  Dimension= "2"
...
Proc
  Name = "PB"
  Dimension= "2"
...
Proc
  Name = "PC"
  Dimension= "2"
...
Mem
```

```

    Name = "M"
    ...
Net
    Name = "N"
    Sender = { PA[0..1], PB[0..1], PC, M }
    Receiver = { PA[0..1], PB[0..1], PC, M }
Morph
    Name = "Y0"
    ConfiguresIngredients = {}
    IntoResources = { M, N }
Morph
    Name = "YA"
    Dimension = "2"
    ConfiguresIngredients = { X[+0] }
    IntoResources = { PA[+0] }
    LatencyFromMorph
        FromMorph = YB[+0]
        Latency = 15
    LatencyFromMorph
        FromMorph = YC
        Latency = 50
Morph
    Name = "YB"
    Dimension = "2"
    ConfiguresIngredients = { X[+0] }
    IntoResources = { PB[+0] }
    LatencyFromMorph
        FromMorph = YA[+0]
        Latency = 10
    LatencyFromMorph
        FromMorph = YC
        Latency = 50
Morph
    Name = "YC"
    ConfiguresIngredients = { X[0..1] }
    IntoResources = { PC }
    LatencyFromMorph
        FromMorph = YA[*]
        Latency = 70
    LatencyFromMorph
        FromMorph = YB[*]
        Latency = 75

```

5. Topology

The topology of an architecture is entirely described by the `Sender` and `Receiver` sets of the network links in that architecture. This section presents certain restrictions on allowable topologies. It is important to note that these restrictions do not ensure that arbitrary code may be executed on an architecture. For instance, an architecture without any processors could meet the topology restrictions. Tools such as a high-level compiler should reject code that cannot be mapped to an architecture.

5.1. Definitions

It is useful to define certain relationships between resources:

Resource *A* is *immediately connected to* a resource *B* if (1) *B* is a network link and *A* is in its `Sender` set, (2) *A* is a network link and *B* is in its `Receiver` set, or (3) *A* and *B* are both network links and *A* is directly connected to *B*. This relationship is unidirectional; *A* directly connected to *B* does not imply *B* directly connected to *A*. These cases are illustrated in Figure 2:

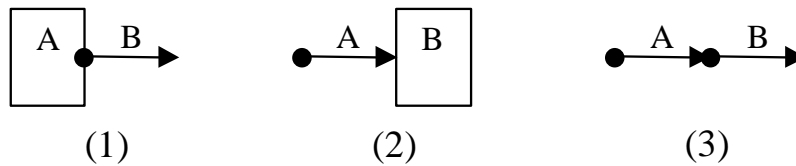


Figure 2. These are the three conditions for which *A* is immediately connected to *B*.

Resource *A* is *connected to* Resource *B* if *A* is immediately connected to *B* or there is a series of network links N_0, N_1, \dots, N_m such that *A* is immediately connected to N_0 , N_i is immediately connected to N_{i+1} and N_m is immediately connected to *B*. For example:

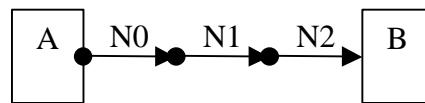


Figure 3. *A* is connected to *B* through a series of network links.

A is said to be connected to *B* through the *route* composed of $N_0 \dots N_m$. Multiple routes may connect *A* to *B*. The *latency* of a route is equal to the sum of latencies of all links involved; the *bandwidth* of a route is equal to the minimum bandwidth of all links involved.

5.2. Processor Topology

A processor may read only from memories connected to it and write only to memories it is connected to. A functional unit of a processor that can load, peek, or pop can only have memories in its `AccessesMem` parameter that are connected to the processor. Similarly, a functional unit of a processor that can store and/or push can have memories

in its `AccessesMem` parameter only if the processor is connected to that memory. A kernel executed by a processor may not have an input stream or block in a memory that is not connected to the processor and may not have an output stream or block in a memory that the processor is not connected to.

5.3. Cache Topology

An architecture must contain a strict hierarchy of caches, in which each cache is backed by one and only one RAM memory. To determine if an architecture has a strict cache hierarchy, each level of caches is first identified in turn. If a cache is connected to a processor, it is an L0 cache. For each successive level i , if a cache is connected to an $L(i-1)$ cache and is not already identified as an L_j cache where $i > j$, then it is an L_i cache. If a cache remains unidentified at the end of this process, the architecture is illegal. An L0 cache must be connected to one or more processors, and must have only one L1 cache or one RAM memory connected to it. An L_i cache where $i > 0$ must be connected to one or more $L(i-1)$ caches, and must have only one $L(i+1)$ or one RAM memory connected to it. A processor also may be connected to an L0 cache that is connected to it, but the L0 cache then must be connected to the L1 cache that is connected to it, and so on until a RAM memory is reached. No other connections involving caches are allowed.

Further, a processor may not be serviced by two caches with the same type of content (instructions or data) backed by the same RAM memory. Similarly, a cache may not be backed by another cache that cannot contain the same type of content.

5.3.1. Cache Topology Example

The following diagram illustrates an allowable hierarchy of processor, cache, and RAM:

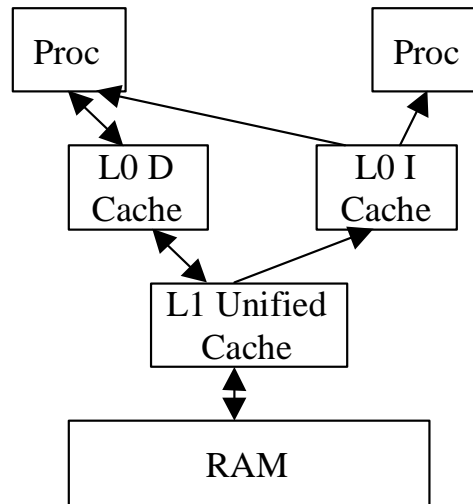


Figure 4. An allowable hierarchy of processor, cache and RAM objects.

6. References

1. “Introduction to Morphware: Software Architecture for Polymorphous Computing Architectures”, Version 1.0, February 23, 2004. Available at www.morphware.org.
2. “The PCA Metadata System”, Georgia Institute of Technology and SPAWAR Systems Center San Diego, March. 2, 2004. Available at www.morphware.org.
3. XML Schema, World Wide Web Consortium, www.w3.org/XML/Schema.
4. M. B. Taylor et al, “The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs,” IEEE Micro, March-April 2002. See also cag-www.lcs.mit.edu/raw/documents.

Appendix: Machine Model XML Schema

This is the XML Schema for the PCA machine model.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>

  <xs:element name="Root">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Name" type="xs:string"/>
        <xs:element name="AddressSize" type="xs:int" minOccurs="0"/>
        <xs:element name="WordSize" type="xs:int" minOccurs="0"/>
        <xs:element name="BigEndian" type="xs:boolean" minOccurs="0"/>
        <xs:element name="AlignmentChar" type="xs:int" minOccurs="0"/>
        <xs:element name="AlignmentShort" type="xs:int" minOccurs="0"/>
        <xs:element name="AlignmentInt" type="xs:int" minOccurs="0"/>
        <xs:element name="AlignmentLong" type="xs:int" minOccurs="0"/>
        <xs:element name="AlignmentFloat" type="xs:int" minOccurs="0"/>
        <xs:element name="AlignmentDouble" type="xs:int" minOccurs="0"/>
        <xs:element name="AlignmentBitfield" type="xs:int" minOccurs="0"/>
        <xs:element name="BitfieldCrossWord" type="xs:boolean" minOccurs="0"/>
        <xs:element name="Ingredient" type="mm_Ingredient"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Proc" type="mm_Proc"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Mem" type="mm_Mem"
          minOccurs="0" maxOccurs="unbounded"/>
        <xs:element name="Net" type="mm_Net"
          minOccurs="1" maxOccurs="unbounded"/>
        <xs:element name="Morph" type="mm_Morph"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="mm_Ingredient">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Dimension" type="Dimension" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:simpleType name="Dimension">
    <xs:list itemType="xs:string"/>
  </xs:simpleType>

  <xs:complexType name="mm_Proc">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Dimension" type="Dimension" minOccurs="0"/>
      <xs:element name="Master" type="mm_Ref"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="SupportsUserCode" type="xs:boolean"/>
      <xs:element name="SupportedKernel" type="mm_Kernel"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="ContextSwitch" type="xs:int" minOccurs="0"/>
      <xs:element name="Multiplexing" type="xs:int" minOccurs="0"/>
      <xs:element name="Freq" type="xs:int" minOccurs="0"/>
      <xs:element name="SupportsChar" type="xs:boolean" minOccurs="0"/>
      <xs:element name="SupportsShort" type="xs:boolean" minOccurs="0"/>
      <xs:element name="SupportsInt" type="xs:boolean" minOccurs="0"/>
      <xs:element name="SupportsLong" type="xs:boolean" minOccurs="0"/>
      <xs:element name="SupportsFloat" type="xs:boolean" minOccurs="0"/>
      <xs:element name="SupportsDouble" type="xs:boolean" minOccurs="0"/>
      <xs:element name="FU" type="mm_FU" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="BranchMispredict" type="xs:int" minOccurs="0"/>
      <xs:element name="NumRegs" type="xs:int" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```

<xs:element name="ISize" type="xs:int" minOccurs="0"/>
<xs:element name="NumSIMDClusters" type="xs:int" minOccurs="0"/>
<xs:element name="SIMDSubword" type="xs:boolean" minOccurs="0"/>
<xs:element name="LLCQuality" type="xs:float" minOccurs="0"/>
<xs:element name="LLCSwPipelineQuality" type="xs:float" minOccurs="0"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="mm_FU">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Dimension" type="Dimension" minOccurs="0"/>

    <xs:element name="LatencyBitwise" type="xs:int" minOccurs="0"/>
    <xs:element name="PipelineBitwise" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CycleNumber" type="xs:int"/>
          <xs:element name="OccupiedStage" type="xs:int"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="LatencySelect" type="xs:int" minOccurs="0"/>
    <xs:element name="PipelineSelect" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CycleNumber" type="xs:int"/>
          <xs:element name="OccupiedStage" type="xs:int"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="LatencyIComp" type="xs:int" minOccurs="0"/>
    <xs:element name="PipelineIComp" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CycleNumber" type="xs:int"/>
          <xs:element name="OccupiedStage" type="xs:int"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="LatencyIAdd" type="xs:int" minOccurs="0"/>
    <xs:element name="PipelineIAdd" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CycleNumber" type="xs:int"/>
          <xs:element name="OccupiedStage" type="xs:int"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="LatencyIMul" type="xs:int" minOccurs="0"/>
    <xs:element name="PipelineIMul" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="CycleNumber" type="xs:int"/>
          <xs:element name="OccupiedStage" type="xs:int"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="LatencyIDiv" type="xs:int" minOccurs="0"/>
    <xs:element name="PipelineIDiv" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>

```

```
<xs:sequence>
  <xs:element name="CycleNumber" type="xs:int"/>
  <xs:element name="OccupiedStage" type="xs:int"
    minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>

<xs:element name="LatencyFPComp" type="xs:int" minOccurs="0"/>
<xs:element name="PipelineFPComp" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CycleNumber" type="xs:int"/>
      <xs:element name="OccupiedStage" type="xs:int"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="LatencyFPAdd" type="xs:int" minOccurs="0"/>
<xs:element name="PipelineFPAdd" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CycleNumber" type="xs:int"/>
      <xs:element name="OccupiedStage" type="xs:int"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="LatencyFPMul" type="xs:int" minOccurs="0"/>
<xs:element name="PipelineFPMul" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CycleNumber" type="xs:int"/>
      <xs:element name="OccupiedStage" type="xs:int"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="LatencyFPDiv" type="xs:int" minOccurs="0"/>
<xs:element name="PipelineFPDiv" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CycleNumber" type="xs:int"/>
      <xs:element name="OccupiedStage" type="xs:int"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="LatencyLoad" type="xs:int" minOccurs="0"/>
<xs:element name="PipelineLoad" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CycleNumber" type="xs:int"/>
      <xs:element name="OccupiedStage" type="xs:int"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="LatencyStore" type="xs:int" minOccurs="0"/>
<xs:element name="PipelineStore" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="CycleNumber" type="xs:int"/>
      <xs:element name="OccupiedStage" type="xs:int"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```

    </xs:complexType>
  </xs:element>

  <xs:element name="LatencyPeek" type="xs:int" minOccurs="0"/>
  <xs:element name="PipelinePeek" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CycleNumber" type="xs:int"/>
        <xs:element name="OccupiedStage" type="xs:int"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="LatencyPop" type="xs:int" minOccurs="0"/>
  <xs:element name="PipelinePop" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CycleNumber" type="xs:int"/>
        <xs:element name="OccupiedStage" type="xs:int"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="LatencyPush" type="xs:int" minOccurs="0"/>
  <xs:element name="PipelinePush" minOccurs="0" maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="CycleNumber" type="xs:int"/>
        <xs:element name="OccupiedStage" type="xs:int"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="AccessesMem" type="mm_Ref"
    minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="mm_Kernel">
  <xs:sequence>
    <xs:element name="Name" type="xs:string"/>
    <xs:element name="Multiplex" type="xs:boolean" minOccurs="0"/>
    <xs:element name="PrologueTime" type="xs:float" minOccurs="0"/>
    <xs:element name="EpilogueTime" type="xs:float" minOccurs="0"/>
    <xs:element name="InitiationInterval" type="xs:float" minOccurs="0"/>
    <xs:element name="Latency" type="xs:float" minOccurs="0"/>
    <xs:element name="StreamInfo" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Name" type="xs:string"/>
          <xs:element name="StoredInMem" type="mm_Ref"/>
          <xs:element name="Size" type="xs:int"/>
          <xs:element name="InputRate" type="xs:int"/>
          <xs:element name="OutputRate" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="BlockInfo" minOccurs="0" maxOccurs="unbounded">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="Name" type="xs:string"/>
          <xs:element name="StoredInMem" type="mm_Ref"/>
          <xs:element name="Size" type="xs:int"/>
          <xs:element name="InputRate" type="xs:int"/>
          <xs:element name="OutputRate" type="xs:int"/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

```

```

    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="mm_Mem">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Dimension" type="Dimension" minOccurs="0"/>
      <xs:element name="Subtype">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="FIFO"/>
            <xs:enumeration value="RAM"/>
            <xs:enumeration value="CACHE"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="Size" type="xs:long"/>
      <xs:element name="FIFOPeek" type="xs:int" minOccurs="0"/>
      <xs:element name="RAMCoherence" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="COHERENT"/>
            <xs:enumeration value="INCOHERENT"/>
            <xs:enumeration value="UNCACHEABLE"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="RAMStreamEfficientSize" type="xs:int" minOccurs="0"/>
      <xs:element name="RAMLinearBw" type="xs:float" minOccurs="0"/>
      <xs:element name="RAMRandomBw" type="xs:float" minOccurs="0"/>
      <xs:element name="RAMInlineBw" type="xs:float" minOccurs="0"/>
      <xs:element name="CacheContent" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="DATA"/>
            <xs:enumeration value="INSTRUCTIONS"/>
            <xs:enumeration value="UNIFIED"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element name="CacheCoherent" type="xs:boolean" minOccurs="0"/>
      <xs:element name="CacheLinesize" type="xs:int" minOccurs="0" maxOccurs="2"/>
      <xs:element name="CacheAssociativity" type="xs:int" minOccurs="0"/>
      <xs:element name="CacheWriteback" type="xs:boolean" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="mm_Net">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Dimension" type="Dimension" minOccurs="0"/>
      <xs:element name="Sender" type="mm_Ref" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Receiver" type="mm_Ref"
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Latency" type="xs:float"/>
      <xs:element name="Bw" type="xs:float"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="mm_Morph">
    <xs:sequence>
      <xs:element name="Name" type="xs:string"/>
      <xs:element name="Dimension" type="Dimension" minOccurs="0"/>
      <xs:element name="ConfiguresIngredient" type="mm_Ref" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="IntoResource" type="mm_Ref" minOccurs="0"
        maxOccurs="unbounded"/>
      <xs:element name="LatencyFromMorph" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="FromMorph" type="mm_Ref"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>

```

```
        <xs:element name="Latency" type="xs:float" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:sequence>
</xs:complexType>

<xs:complexType name="mm_Ref">
  <xs:sequence>
    <xs:element name="Name" type="xs:string" />
    <xs:element name="Index" type="IndexType" minOccurs="0" />
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="IndexType">
  <xs:list itemType="xs:string" />
</xs:simpleType>
</xs:schema>
```