Enhancing Domain Specific Language Implementations Through Ontology

WOLFHPC15: Fifth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing

Nov. 16, 2015

Chunhua "Leo" Liao, Pei-Hung Lin , Daniel J. Quinlan,
Yue Zhao* , and Xipeng Shen*

LLNL-PRES-679077

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

* NC STATE

Lawrence Livermore National Laboratory

Outline

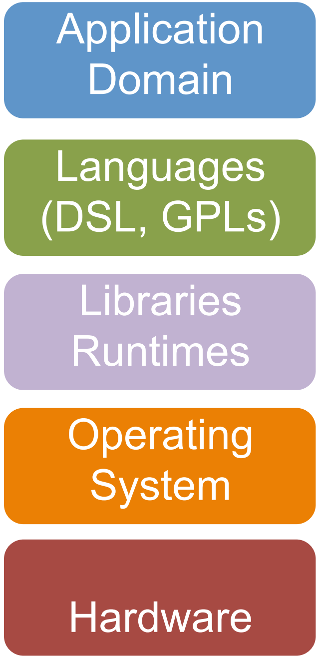
- Motivation
- Methodology
  - Ontology-based knowledge base
  - Compiler/runtime interface
- Evaluation
  - Ontology for Stencil computation
  - Compiler implementation
  - Preliminary results
- Related work
- Discussion

Motivation

- DSLs: attractive to program HPC machines
  - High-level annotations with rich semantics to efficiently express domain algorithms
  - Performance: DSL implementations (compilers + runtime systems)
- Semantic gap: Large body of knowledge is required to build efficient implementations
  - Application domain optimization knowledge
  - Language semantics: DSLs and host languages
  - Library interface semantics
  - Hardware architecture features
- Current problem: all the knowledge is implicitly assumed or represented using ad-hoc approaches
  - informal, not reusable, not scalable
- Our solution: adapt modern knowledge-engineering method to enhance DSL implementations

What knowledge?

- Application domains
  - Data: grid, points, stencil, halo, …
  - Algorithms: Iterative algorithm, convergence
- Language semantics: general-purpose languages and DSLs
  - non-aliasing, non-overlapping, fixed-sized vs resizable
  - functions : read/write variable sets, pure, virtual
- Library knowledge: standard and domain-specific
  - Containers: unique, ordered, sorted, continuous storage
  - Iterators: random access, bidirectional
- HPC hardware architecture configurations
  - CPUs: number of cores, cache sizes
  - NVIDIA GPU: shared memory, global memory, constant memory



HPC software/hardware stack

DSL knowledge is often hard to extract, critical to performance

- High-level abstractions in DSL
  - Encouraging code reuse and hiding implementation from interfaces to reduce software complexity
  - Functions, data structures, classes and templates …
  - Could be standard or user-defined/domain-specific

- Semantics of high-level abstractions
  - Critical to optimizations, including parallelization
    - Read-only semantics
  - Hard to be extracted by static analysis
    - STL vector implementation ?→? elements are contiguous in memory

- Conventional compilers lose track of abstractions
  - Analyses and optimizations are mostly done on top of middle or low level IR
  - Hard to trace back to the high-level abstractions represented in source level

Problems with current HPC knowledge management approaches

- Informal and Ad-Hoc
- Isolated: separated managing software and hardware knowledge
- Hard to reusable, not scalable
- No toolchain support:
  - Parser: each solution has its own parser
  - Lack of IDE, validation tools, etc.
- Not easy to share among DSL designers, domain experts and DSL developers

```
class std::vector<T> {
  alias none; overlap none; //elements are alias-free and non-overlapping
  is_fixed_sized_array { //semantic-preserving functions as a fixed-sized array
    length(i) = {this.size()};
    element(i) = {this.operator[](i); this.at(i);};
  };
};
void my_processing(SgNode* func_def) {
  read{func_def}; modify {func_def};  //side effects of a function
}
std::list<SgFunctionDef*> findCFunctionDefinition(SgNode* root){
  read {root};  modify {result};
  return unique; //return a unique set
}
void Compass::OutputObject::addOutput(SgNode* node){
   read {node};
   //order-independent side effects
   modify {Compass::OutputObject::outputList<order_independent>};
}
```

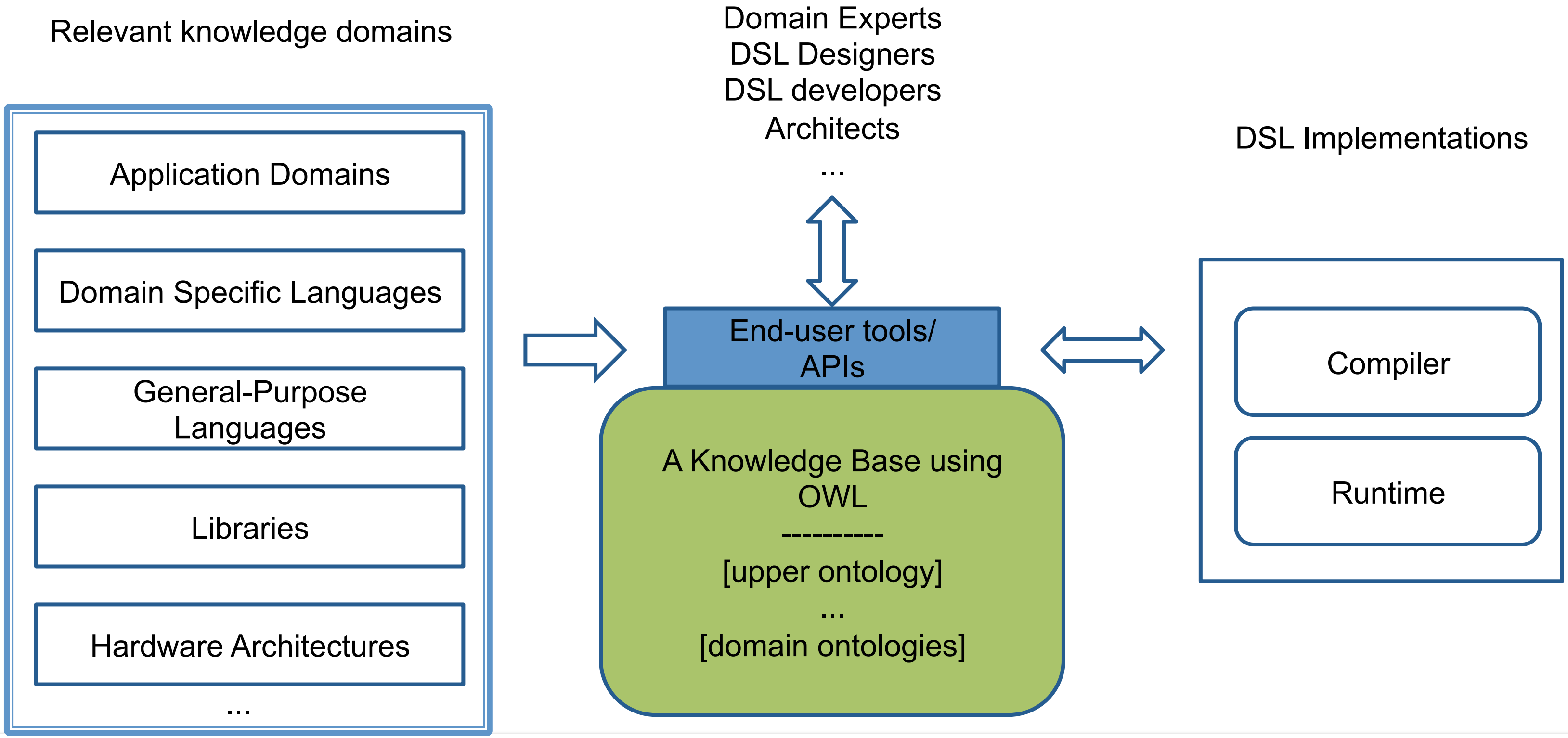
Semantic specification file Liao'10

Mem spec of Tesla M2075:

```
die =1 tpc; tpc = 16 sm; sm = 32 core;
globalMem 8 Y rw na 5375M 128B ? 600clk <L2 L1> <> die <0.1 0.5> warp{⌊address1/blockSize⌋ != ⌊address2/blockSize⌋};
L1 9 N rw na 16K 128B ? 80clk <> <L2 globalMem> sm ? warp{⌊address1/blockSize⌋ != ⌊address2/blockSize⌋};
L2  7 N rw na  768K 32B ? 390clk om om die ? warp{⌊address1/blockSize⌋ != ⌊address2/blockSize⌋};

constantMem 1 Y r na 64K ? ? 360clk <cL2 cL1> <> die ? warp{address1 != address2};
cL1 3 N r na 4K 64B ? 48clk <> <cL2 constantMem> sm ? warp{⌊address1/blockSize⌋ != ⌊address2/blockSize⌋};
cL2 2 N r na 32K 256B ? 140clk <cL1> <cL2 constantMem> die ? warp{⌊address1/blockSize⌋ != ⌊address2/blockSize⌋};

sharedMem 4 Y rw na 48K ? 32 48clk <> <>  sm ? block{word1!=word2 && word1%banks ==word2%banks};

 ...   ...
```

Memory Specification Language: Chen'14

A new ontology-driven DSL implementation paradigm

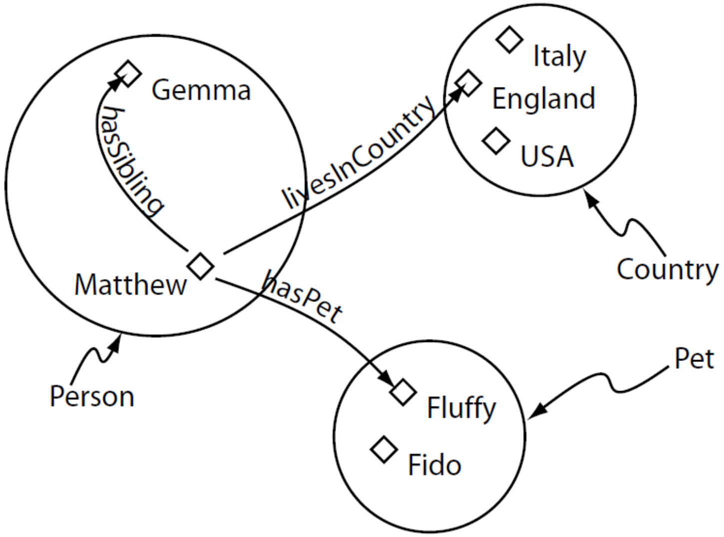


Central component: ontology-based knowledge base

- Definitions:
  - Philosophy: the study of nature of beings and their relations
  - Modern: a formal specification for explicitly representing knowledge of the entities in a domain

- Provides a common vocabulary in a domain
  - Concepts, properties, and individuals

- Theory foundation: description logics (DLs), a family of logical languages for knowledge representation
  - several dialects with different expressiveness and efficiencies (for reasoning)

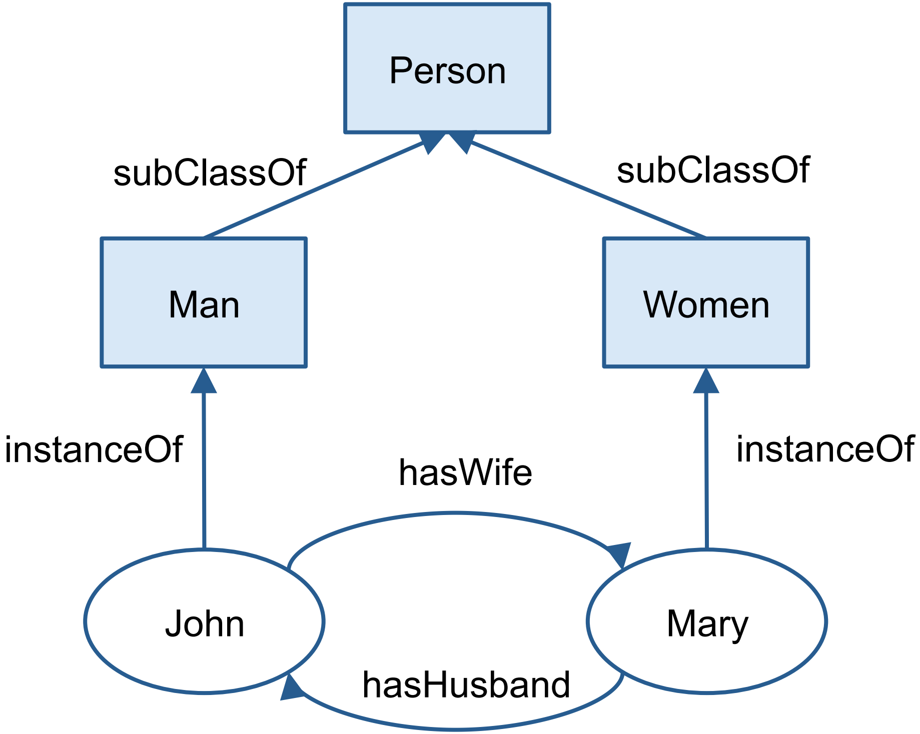


Web Ontology Language

- Web Ontology Language (OWL): the most popular ontology languages
  - Classes (or concepts): denote sets of individuals. organized in a hierarchy
  - Individuals (or instances): single instances in the domain
  - Properties (or relations): binary relations between entities
  - Maturing tool chain since ’04 by W3C: Protégé IDE, OWL API, FaCT++, SWI-Prolog, etc.

Functional Syntax	Formal Semantics	Natual Language Semantics
$Declaration(Class(CE))$	$(CE)^C \subseteq \Delta_I$	CE is a class within an object domain
$Declaration(NamedIndividual(a))$	$(a)^I \in \Delta_I$	a is an individual within an object domain
$Declaration(ObjectProperty(OPE))$	$(OPE)^{OP} \subseteq \Delta_I \times \Delta_I$	OPE is an object property connecting two objects
$subClassOf(CE_1\ CE_2)$	$(CE1)^C \subseteq (CE2)^C$	class CE_1 is a subclass of class CE_2
$ClassAssertion(CE\ a)$	$(a)^I \in (CE)^C$	individual a is an instance of class CE
$ObjectPropertyAssertion(OPE\ a_1\ a_2)$	$((a_1)^I, (a_2)^I) \in (OPE)^{OP}$	a_1 is related to a_2 via ObjectProperty OPE
$ObjectIntersectionOf(CE1\ ...\ CEn)$	$(CE_1)^C \cap ... \cap (CE_n)^C$	a class resulting from intersecting class CE_1 to CE_n

Example family ontology using OWL

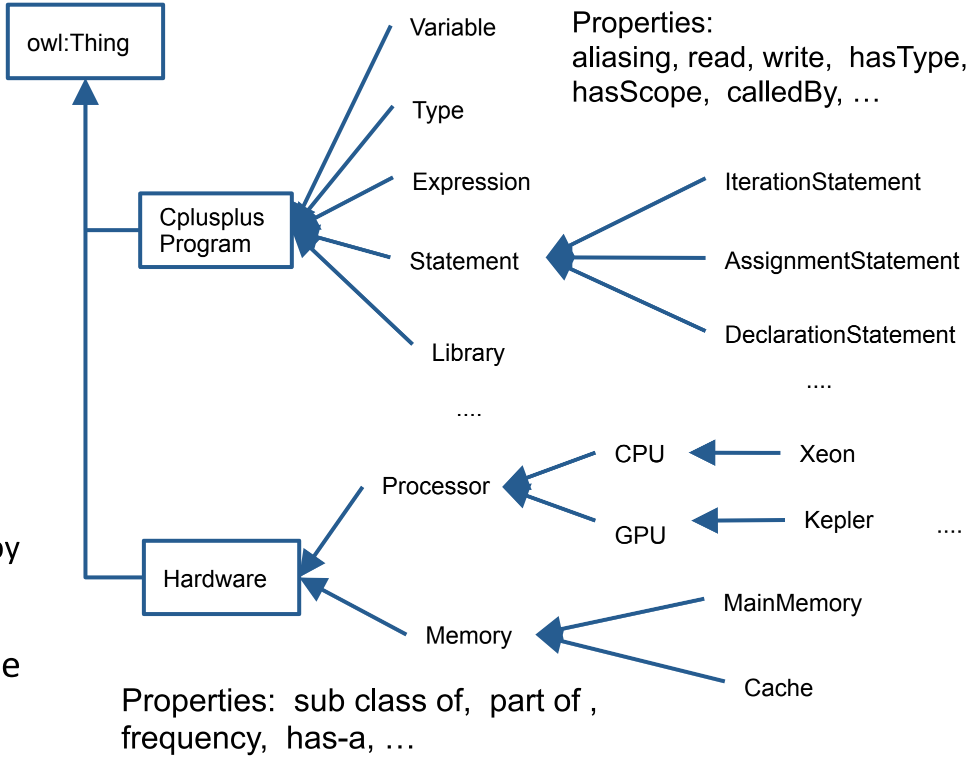



```
Prefix(:=<http://example.com/owl/families/>)
Ontology(<http://example.com/owl/families>
 Declaration( NamedIndividual( :John ) )
 Declaration( NamedIndividual( :Mary ) )
 Declaration( Class( :Person ) )
 Declaration( Class( :Woman ) )
 Declaration( ObjectProperty( :hasWife ) )
 Declaration( ObjectProperty( :hasSpouse) )
 SubClassOf( :Woman :Person )
 SubClassOf( :Man  :Person )
 ClassAssertion(:Man :John)
 ClassAssertion(:Women :Mary)
 SubObjectPropertyOf( :hasWife :hasSpouse )
 ObjectPropertyAssertion( :hasWife :John :Mary )
 ... )
```

Family ontology in functional syntax

Software and hardware ontology for DSL implementations

- Requirements:
  - common vocabulary
  - intuitive, efficient to facilitate DSL analyses and optimizations
- Software: based on C and C++ language standards
  - Individuals: use international resource identifiers (IRIs) – "http://example.com/owl/CProgram:Type"
    - Scoped IRI - qualified name + relative location
    - E.g.: file.c::foo()::1:6,1:10 std::vector<T>, std::vector<int>,
- Hardware: focus on single machine for now
  - Individuals: machine1, cpu2, memory , identified by names or serial numbers
- Space Efficiency - core knowledge base + loadable supplemental modules (for individuals)



Excerpt of Software and Hardware Ontology with Individuals

```
% program concepts and their hierarchy
Class(:PointerType) SubClassOf(:PointerType :DerivedType)

Class(:BinaryOp) Class(:AddOp)
SubClassOf(:AddOp :BinaryOp)

SubClassOf(:SelectionStatement :Statement)
SubClassOf(:IfStatement :SelectionStatement)

% define the relations between program constructs
ObjectProperty(:hasScope) ObjectProperty(:hasType)
ObjectProperty( :alias)  ObjectProperty(:read)

%  variables identified by a qualified names
NamedIndividual ( :var1)  NamedIndividual ( :var2)

% two pointer variables alias each other
ObjectPropertyAssertion(:hasType :var1 :PointerType) …
ObjectPropertyAssertion (:alias :var1  :var2)
```

```
% hardware concepts and hierarchy
Class(:Xeon)  Class(:CPU)
SubClassOf(:Xeon :CPU )
Class( :X5680 )
SubClassOf( :X5680 :Xeon )
ClassAssertion(:X5680 :cpu1)

Class(:GPU)
SubClassOf(:Quadro_4000 :GPU)
SubClassOf(:Quadro_4000 DataHasValue(:numberOfCores "256"))

% Individuals
NamedIndividual(:gpu1)
NamedIndividual(:cpu1)
ClassAssertion(:Quadro_4000 :gpu1)

% a workstation with CPU and GPU
NamedIndividual(:tux322)
ClassAssertion(:computer :tux322)
ObjectPropertyAssertion(:hasPart :tux322 :gpu1)
ObjectPropertyAssertion(:hasPart :tux322 :cpu2)
```

Compiler and runtime interface of ontology-based knowledge base

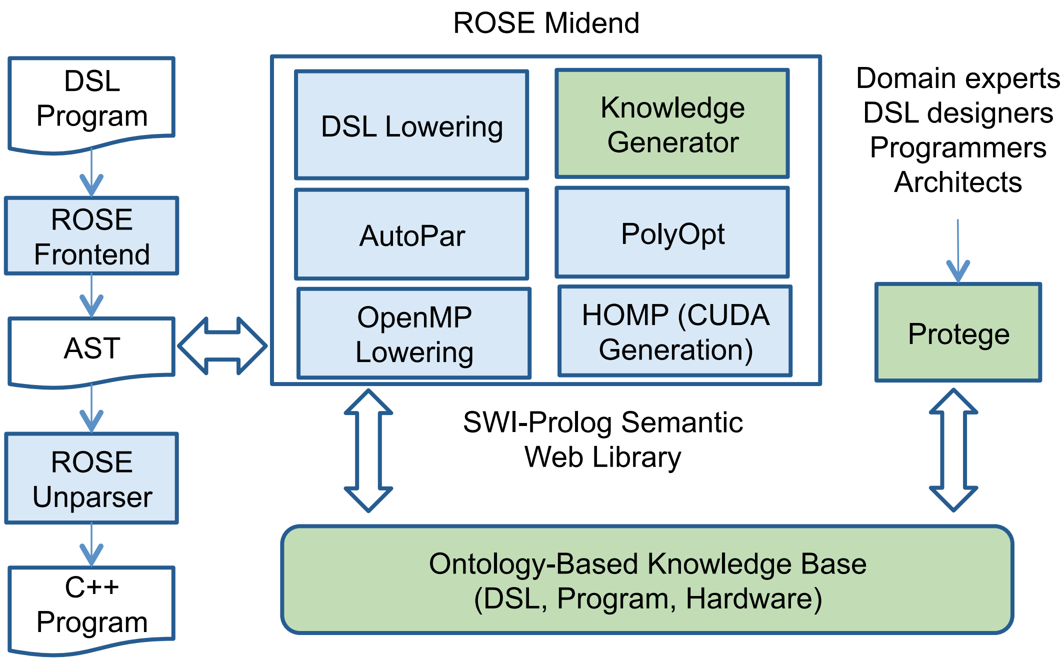
- Requirements
  - Bidirectional: query + insertion
  - Support both on-disk and in-memory storage
- Knowledge representation: OWL
  - On disk as OWL files
  - In-memory storage: SWI-Prolog predicates
- Two kinds of interfaces
  - Prolog reasoning engine: powerful declarative Prolog queries operating on in-memory storage
    - Learning curve
  - Prebuilt C/C++ query interface: light weight operating on OWL files, easy for deployment

```
% Prolog query interfaces
% Define additional predicates
checkClass(IndividualL, ClassL) :-
        rdf(IndividualL, rdf:type, ClassL).
% recursively check through subclasses
checkClass(IndividualL, ClassL) :-
        rdf(SubClassL, rdfs:subClassOf, ClassL),
        checkClass(IndividualL, SubClassL).
% if individual tux322 is a computer?
?- checkClass ('tux322', ':computer')
% What are the variables written by function bar?
?- Function(X),hasName(X,'bar'),writtenBy(Vars,X)
```

```
// C++ query interface

std::string getMachineName (string machine_IRI);
vector<std::string> getCPUs (string machine_name);
vector<std::string> getGPUs (string machine_name);
int getCoreCount (string gpu_IRI);
```

Evaluation benefits of ontology for DSL implementations

- The DSL: Shift Calculus
  - Light-weight embedded DSL
  - Host language: C++
  - Leverage Chombo AMR library [Colella'09]
- The implementation: a ShiftCalculus source-to-source translator
  - Built on top of ROSE, a source-to-source compiler infrastructure developed at LLNL
  - Connecting to ontology



Ontology-based knowledge base for DSL implementations

ShiftCalculus DSL syntax and semantics

Mathematical Object	Mathematical Notation	Notes, including syntax in C++11 reference implementation if appropriate.
Point on grid	$\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{s} \dots \in \mathbb{Z}^D$	`Point`
Sets of gridpoints	$\Lambda, \Omega, \Gamma \subset \mathbb{Z}^D$	`Box` (Rectangular sets only)
Arrays over rectangular grids (RectMDArrays)	$\phi : \Omega \rightarrow \mathbb{R}$ $U : \Lambda \rightarrow \mathbb{R}^N \times \mathbb{R}^D$	`RectMDArray<double> phi(Omega);` `RectMDArray<double,N,DIM> U(Lambda);`
Shift Operators	$\mathcal{S}^{\boldsymbol{p}}(\Lambda) \equiv \Lambda + \boldsymbol{p}$ $(\mathcal{S}^{\boldsymbol{p}}(U))_{\boldsymbol{i}} \equiv U_{\boldsymbol{i}+\boldsymbol{p}}$	`Shift S = Shift::getUnitShift();`

Single-level stencil operators	$\mathcal{L} = \sum\limits_{\boldsymbol{s}} a_{\boldsymbol{s}} \mathcal{S}^{\boldsymbol{s}}$
	$\mathcal{L}(\phi)_{\boldsymbol{i}} = \sum\limits_{\boldsymbol{s}} a_{\boldsymbol{s}} \phi_{\boldsymbol{i}+\boldsymbol{s}}, \boldsymbol{i} \in \Gamma$
		`Point s_0 = ...;double a_0 = ...; ...;`
		`Stencil<double> L=a_0*(S^s_0)+a_1*(S^s_1)+...;`
		`RectMDArray<double> LOfPhi = L(phi,Gamma);`

Pointwise application of functions	$f : \mathbb{R}^N \times \mathbb{R}^D \rightarrow \mathbb{R}$ $(f@(U)$ **on** $\Gamma) : \Gamma \rightarrow \mathbb{R}$ $(f@(U))_{\boldsymbol{i}} \equiv f(U_{\boldsymbol{i}}), \boldsymbol{i} \in \Gamma$	`forall(U,FOfU,F,Gamma)` applies the function .

ShiftCalculus DSL for stencil computation

```
#define DIM 3
int main(int argc, char* argv[])
  {
    Point lo, hi;
  // Space discretization
    Box bxdest(lo,hi);
    Box bxsrc=bxdest.grow(1);
...
  // Source and destination data containers
    RectMDArray<double,1> Asrc(bxsrc);
    RectMDArray<double,1> Adest(bxdest);
...
    double ident, C0;
  // Shift and Stencil declarations
    array<Shift,DIM> shft_vec = getShiftVec();
    Stencil<double> laplace = C0*(shft_vec^zero);
  // Stencil formation using Shift
    for (int dir=0;dir<DIM;dir++)
      {
        Point thishft = getUnitv(dir);
        laplace = laplace + ident*(shft_vec^thishft);
        laplace = laplace + ident*(shft_vec^(thishft*(-1)));
      }
   // Apply stencil computation using data containers in space
    Stencil<double>::apply(laplace, Asrc, Adest, bxdest);
  }
```

3-D 7-point stencil

Constructing stencil at the initial point (red cell in the figure). CO is the coefficient applied to this stencil cell.

Iteratively constructing stencil for the surrounding points (green cells in figure). “ident” is the coefficient applied to these stencil cells.

Applying the stencil, source data, destination data, and destination box to the computation. Destination box represents the loops in the computation.

Generated sequential C++ output for Laplacian example

```
int main (int argc , char * argv [])
{
...
const class Point lo( zero );
const class Point hi = getOnes () * adjustedBlockSize ;
const class Box bxdest (lo ,hi);
const class Box bxsrc = bxdest . grow (1) ;
class RectMDArray < double , 1 , 1 , 1 > Asrc ( bxsrc );
class RectMDArray < double , 1 , 1 , 1 > Adest ( bxdest );
const double ident = 1.0;
const double C0 = -6.00000;
...
double * sourceDataPointer = Asrc.getPointer ();
double * destinationDataPointer = Adest.getPointer ();
for (k = lb2; k < ub2; ++k) {
    for (j = lb1; j < ub1; ++j) {
        for (i = lb0; i < ub0; ++i) {
                destinationDataPointer [ arraySize_X * ( arraySize_Y * k + j) + i] =
                sourceDataPointer [ arraySize_X * ( arraySize_Y * (k + -1) + j)+ i] +
                sourceDataPointer [ arraySize_X * ( arraySize_Y * (k + 1)+ j) + i] +
                sourceDataPointer [ arraySize_X * ( arraySize_Y * k+ (j + -1)) + i] +
                sourceDataPointer [ arraySize_X * (arraySize_Y * k + (j + 1)) + i] +
                sourceDataPointer [ arraySize_X * ( arraySize_Y * k + j) + (i + -1)] +
                sourceDataPointer [ arraySize_X * ( arraySize_Y * k + j) + (i +1)] +
                sourceDataPointer [ arraySize_X * ( arraySize_Y * k + j) +i] * -6.00000;
    …
    } } } }
```

More efficient code generation (parallelization, vectorization, loop tiling, GPU porting, etc.) need:

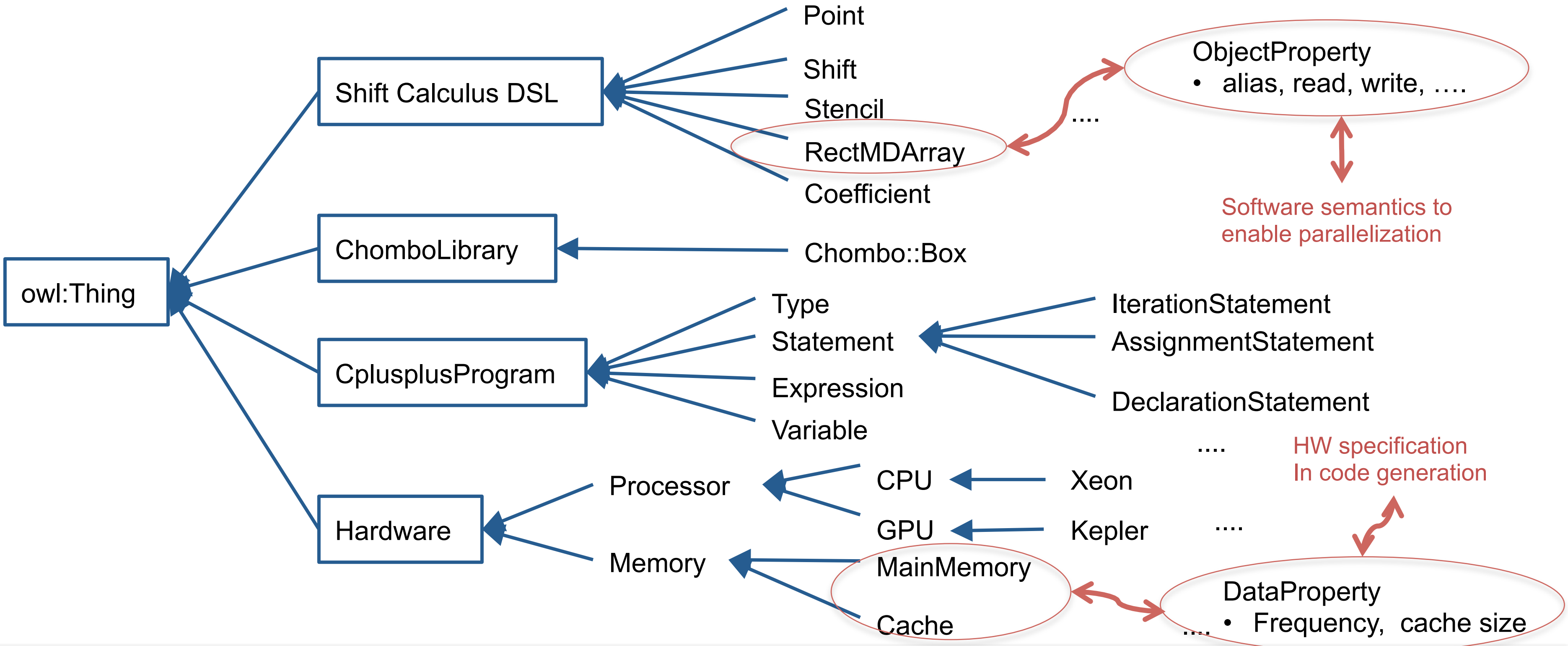
1. Semantics of high level constructs

double * RectMDArray<>::getPointer()

2. Hardware details for target platforms

CPU, GPU, cache, memory size, etc.

Connection between Ontology and DSL implementations



More details about compiler interactions with ontology

- Knowledge generation: traverse AST to generate a program's concrete individuals and relation instances
  - Individual language constructs:
    - functions, loops, types, variable references, statements, function calls, etc.
  - Relations:
    - Structural: hasName, hasParent, hasScope, calledBy, returnedBy ….
    - Analysis: alias, access (read, write), overlap, dependent (true, anti, output) , …

- Semantics propagation
  - Across different levels of IRs: transformation tracking
    - A special model: no memory reuse for AST nodes to ensure unique IDs
    - API functions to store input nodes for a generated node: m: n mapping
    - Update knowledge base with subClassOf(:low-level-entity :high-level-entity) to connect these entities
    - Queries on low-level entities can return semantics associated with high-level entities.
  - Function calls: a = foo(b): read/write formal parameters → actual parameters

Preliminary results

Output Variants	Compiler/ Optimizations	Execution time (sec.)
C++ serial	Baseline	1.51482
C++ OpenMP Parallel	AutoPar	0.380562
C++ Polyhedral Tiled+parallel	PolyOpt	0.503307
CUDA w/ data transfer	HOMP	9.29446
CUDA w/o data transfer	HOMP	3.14713e-05

- Configuration of testing platform:
  - 24-core workstation with Intel Xeon CPU E5-2620 V.3 and 64 GB memory.
  - GCC version 4.8.3, NVCC compiler version 7.0.

Related work

- Ontology-based knowledge bases
  - General purpose: Cyc[Matuszek'06], SUMO[Pease'02]
  - Domain-specific: Gene ontology[Ashburner'00] (Biology), KnowRob [Tenorth'09](service robots), Human behaviors[Rodriguez'14]

- Knowledge engineering methodologies applied to DSL analysis and design
  - Ontology-based domain analysis: [Tairas'09],
  - Translating Ontology to DSL grammars: [Ceh'11]
  - Domain-specific modeling languages: [Brauer'08], [Walter'09]

- Our work : first to use ontology to enhance DSL implementations

Discussion

- A novel ontology-based knowledge base to enhance DSL implementations
  - Enable easy knowledge accumulation, reuse and sharing among software and human users
  - Formal, explicit and uniform format using OWL
  - Initial concepts and relations modeled for software and hardware domains
  - Bidirectional connection to compilers
  - Prototype ShiftCalculus DSL implementation enabled by ontology

- Ongoing/Future work:
  - Collecting and representing stencil optimization knowledge
  - Context-aware optimization advisors
  - Declarative, generic program analysis and transformation

Thank You!

Questions and comments?

Lawrence Livermore National Laboratory

Tiled & parallelized (w/ OMP and SIMD) C++ output

```
Int main(){
…
{    int c0;    int c3;    int c4;    int c5;    int c1;    int c2;
  if (lb0 <= ub0 + -1 && lb1 <= ub1 + -1 && lb2 <= ub2 + -1) {
#pragma omp parallel for private(c2, c1, c5, c4, c3)
    for (c0 = ((lb2 + -31) * 32 < 0?-(-(lb2 + -31) / 32) : ((32 < 0?(-(lb2 + -31) + - 32 - 1) / - 32 : (lb2 + -31 + 32 - 1) / 32))); c0 <= (((ub2 + -1) * 32 < 0?((32 < 0?-((-(ub2 + -1) + 32 + 1) / 32) : -((-(ub2 + -1) + 32 - 1) / 32))) : (ub2 + -1) / 32)); c0++) {
      for (c1 = ((lb1 + -31) * 32 < 0?-(-(lb1 + -31) / 32) : ((32 < 0?(-(lb1 + -31) + - 32 - 1) / - 32 : (lb1 + -31 + 32 - 1) / 32))); c1 <= (((ub1 + -1) * 32 < 0?((32 < 0?-((-(ub1 + -1) + 32 + 1) / 32) : -((-(ub1 + -1) + 32 - 1) / 32))) : (ub1 + -1) / 32)); c1++) {
        for (c2 = ((lb0 + -17) * 18 < 0?-(-(lb0 + -17) / 18) : ((18 < 0?(-(lb0 + -17) + - 18 - 1) / - 18 : (lb0 + -17 + 18 - 1) / 18))); c2 <= (((ub0 + -1) * 18 < 0?((18 < 0?-((-(ub0 + -1) + 18 + 1) / 18) : -((-(ub0 + -1) + 18 - 1) / 18))) : (ub0 + -1) / 18)); c2++) {
          for (c3 = (32 * c0 > lb2?32 * c0 : lb2); c3 <= ((32 * c0 + 31 < ub2 + -1?32 * c0 + 31 : ub2 + -1)); c3++) {
            for (c4 = (32 * c1 > lb1?32 * c1 : lb1); c4 <= ((32 * c1 + 31 < ub1 + -1?32 * c1 + 31 : ub1 + -1)); c4++) {
#pragma ivdep
#pragma vector always
#pragma simd
              for (c5 = (18 * c2 > lb0?18 * c2 : lb0); c5 <= ((18 * c2 + 17 < ub0 + -1?18 * c2 + 17 : ub0 + -1)); c5++) {/* tiled for 3 dims */
destinationDataPointer[c3][c4][c5] = sourceDataPointer[c3 + - 1][c4][c5] + sourceDataPointer[c3 + 1][c4][c5] + sourceDataPointer[c3][c4 + - 1][c5] + sourceDataPointer[c3][c4 + 1][c5] + sourceDataPointer[c3][c4][c5 + - 1] + sourceDataPointer[c3][c4][c5 + 1] + sourceDataPointer[c3][c4][c5] * - 6.00000;
…          }          }          }        }       }     }   }  } }
```

Generated CUDA output

HOST code:

```
int main(int argc,char *argv[])
{…
{
    double *_dev_sourceDataPointer;
    int _dev_sourceDataPointer_size = sizeof(double ) * (1764 - 0);
    _dev_sourceDataPointer = ((double *)(xomp_deviceMalloc(_dev_sourceDataPointer_size)));
    xomp_memcpyHostToDevice(((void *)_dev_sourceDataPointer),((const void
*)sourceDataPointer),_dev_sourceDataPointer_size);
    double *_dev_destinationDataPointer;
    int _dev_destinationDataPointer_size = sizeof(double ) * (1764 - 0);
    _dev_destinationDataPointer = ((double *)
(xomp_deviceMalloc(_dev_destinationDataPointer_size)));
    int _threads_per_block_ = xomp_get_maxThreadsPerBlock();
    int _num_blocks_ = xomp_get_max1DBlock(__final_total_iters__3__ - 1 - 0 + 1);

OUT__1__1527__<<<_num_blocks_,_threads_per_block_>>>(__final_total_iters__3__,__k_interval
__4__,__j_interval__5__,_dev_sourceDataPointer,_dev_destinationDataPointer);
    xomp_freeDevice(_dev_sourceDataPointer);
    xomp_memcpyDeviceToHost(((void *)destinationDataPointer),((const void
*)_dev_destinationDataPointer),_dev_destinationDataPointer_size);
    xomp_freeDevice(_dev_destinationDataPointer);
  }
}
```

Device code:

```
extern "C" __global__ void OUT__1__1527__(int __final_total_iters__3__,int __k_interval__4__,int
__j_interval__5__,double *_dev_sourceDataPointer,double *_dev_destinationDataPointer)
{
 int _p_k;   int _p_j;   int _p_i;   int _p___collapsed_index__7__;
 int _dev_lower;   int _dev_upper;
 int _dev_loop_chunk_size;   int _dev_loop_sched_index;  int _dev_loop_stride;
 int _dev_thread_num = getCUDABlockThreadCount(1);
 int _dev_thread_id = getLoopIndexFromCUDAVariables(1);
 XOMP_static_sched_init(0,__final_total_iters__3__ -
1,1,1,_dev_thread_num,_dev_thread_id,&_dev_loop_chunk_size,&_dev_loop_sched_index,&_dev_loop_stride);
 while(XOMP_static_sched_next(&_dev_loop_sched_index,__final_total_iters__3__ -
1,1,_dev_loop_stride,_dev_loop_chunk_size,_dev_thread_num,_dev_thread_id,&_dev_lower,&_dev_upper))
  for (_p___collapsed_index__7__ = _dev_lower; _p___collapsed_index__7__ <= _dev_upper; _p___collapsed_index__7__
+= 1) {
   _p_k = _p___collapsed_index__7__ / __k_interval__4__ * 1 + lb2;
   int __k_remainder = _p___collapsed_index__7__ % __k_interval__4__;
   _p_j = __k_remainder / __j_interval__5__ * 1 + lb1;
   _p_i = __k_remainder % __j_interval__5__ * 1 + lb0;
    _dev_destinationDataPointer[arraySize_X * (arraySize_Y * _p_k + _p_j) + _p_i] = _dev_sourceDataPointer[arraySize_X *
(arraySize_Y * (_p_k + -1) + _p_j) + _p_i] + _dev_sourceDataPointer[arraySize_X * (arraySize_Y * (_p_k + 1) + _p_j) + _p_i] +
_dev_sourceDataPointer[arraySize_X * (arraySize_Y * _p_k + (_p_j + -1)) + _p_i] + _dev_sourceDataPointer[arraySize_X *
(arraySize_Y * _p_k + (_p_j + 1)) + _p_i] + _dev_sourceDataPointer[arraySize_X * (arraySize_Y * _p_k + _p_j) + (_p_i + -1)] +
_dev_sourceDataPointer[arraySize_X * (arraySize_Y * _p_k + _p_j) + (_p_i + 1)] + _dev_sourceDataPointer[arraySize_X *
(arraySize_Y * _p_k + _p_j) + _p_i] * -6.00000;
  }  }
```