AutoPar: Semantics-Aware Automatic Insertion of OpenMP Directives

Chunhua "Leo" Liao

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

Motivation

- Future extreme-scale architectures will contain computation nodes with abundant parallelism provided by heterogeneous components (such as GPUs) in order to meet performance goals within power constraints.
  - ❑ Many existing High-Performance Computing (HPC) applications exploit only coarse-grain parallelism via MPI, leaving tens of thousands of serial inner loops. Very few of them support fine-grain threading exploiting GPUs.
- Existing parallelization tools mostly focus on Fortran or C applications.
  - ❑ Many LLNL applications are written in C++ with high-level abstractions represented as complex types : classes and templates …
  - ❑ Rich semantics (meanings) are associated with these abstractions
    - *a->foo(x) :* x is **read only** by a->foo()
    - *STL::vector<T>* : elements **stored contiguously**
    - *Loop using iterators:* semantically equal to a classic for loop using an integer loop variable
  - ❑ Traditional tools depending on conventional compilers using low level internal representation (IR)
    - ❑ Difficult to discover high-level abstractions
    - ❑ Even more challenging to extract/leverage associated semantics

Approach

- Recognize high-level abstractions (complex C++ classes, templates, etc.) from Abstract Syntax Tree generated by the ROSE source-to-source compiler
- Encode application semantics via specification files
- Extend classic parallelization algorithms to exploit application semantics

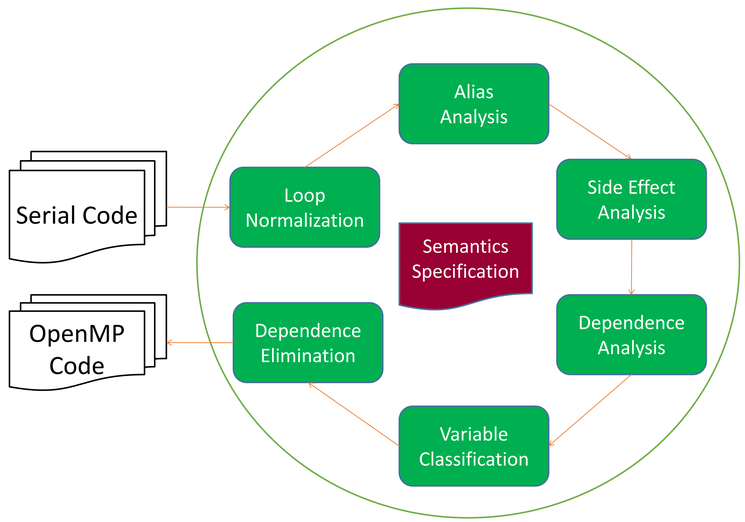
```
class  floatArray {     // user defined array abstraction
  alias none; overlap none; //elements are alias-free and non-overlapping
  is_fixed_sized_array { //semantic-preserving functions as a fixed-sized array
    length(i) = {this.size()};  // array semantics: obtain length
    element(i) = {this.operator[](i); this.elem(i);}; // array element access semantics
  };
};

std::list<SgFunctionDef*> findCFunctionDefinition(SgNode* root){
  read {root};  modify {result}; //side effects of a function
  return unique; //return a unique set
}

operator pow(double val1, double val2)
   {
    modify none; read {val1, val2}; alias none;
   }
```

An example semantics specification file

Details of Semantics-Aware Automatic Parallelization



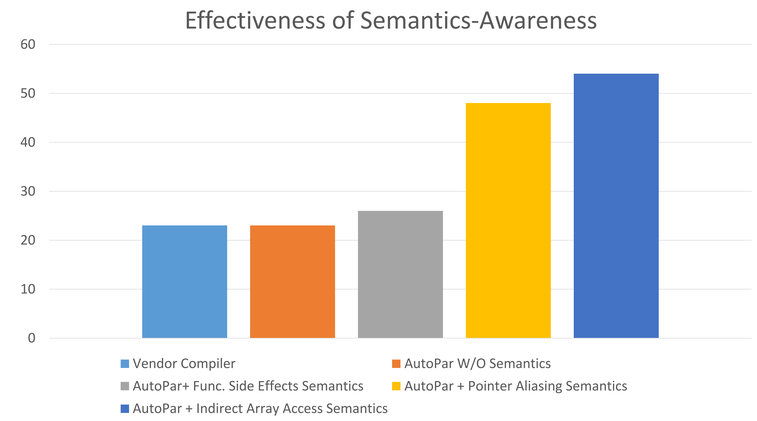
Before	After

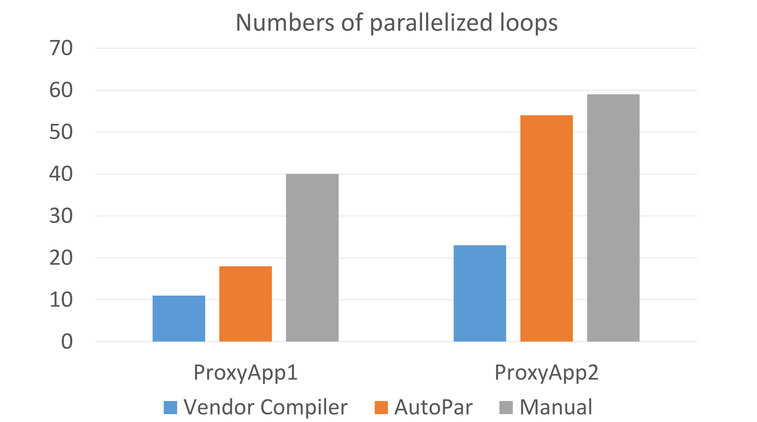
void interpolate1D(
  class floatArray &fineGrid,
  class floatArray &coarseGrid)
{
  …
  for (i = 1; i < _var_0; i += 1) {
  fineGrid.elem(i) =fineGrid.elem(i)+1;
  }
}
	void interpolate1D(class floatArray
	&fineGrid,class floatArray &coarseGrid)
	{ ..
	#pragma omp parallel for private (i)
	firstprivate (_var_0)
	  for (i = 1; i <= _var_0 - 1; i += 1) {
	  fineGrid.elem (i) = fineGrid. elem (i) + 1;
	  }
	}

Additional features requested while working with LLNL application teams:

1. Undo loop normalization: users want their loops unchanged.
2. A helper tool to move variable declarations into innermost scopes: reducing the number of shared variables passed around
3. Generate patches instead of outputting lots of files with scattered changes
4. Support checking correctness of existing OpenMP directives
5. Verify correctness of generated OpenMP codes: using third party tools like Intel Inspector to catch data races. User-provided semantics can be wrong.

Results





Ongoing and Future Work

- Standardize the semantics representation: ontology-based formats (OWL, JSON-LD)
- Support linearized array access: a[c1*i+c2*j + c3], often subscript terms are calculated separately in advance
- Incorporate profitability analysis: parallelizable → worth parallelizing
- Generate OpenMP 4.x directives for GPUs
  More info: http://rosecompiler.org/ROSE_HTML_Reference/auto_par.html

IM-POST-736562