PATUS Quickstart

PATUS v.0.1

Matthias-M. Christen
University of Lugano, Switzerland
matthias.christen@usi.ch

November 2, 2012

1 Introduction

PATUS is a code generation and auto-tuning tool for the class of stencil computations.

Stencil computations are constituent computational building blocks in many scientific and engineering codes. These codes typically achieve a low fraction of peak performance. Computational domains that involve stencils include medical and life science applications, petroleum reservoir simulations, weather and climate modeling, and physics simulations such as fluid dynamics or quantum chromodynamics. Stencil codes may perform tens of thousands of iterations over the spatial domain in order to resolve the time-dependent solution accurately; hence, may require significant core hours on supercomputers. Thus, any performance improvement may lead to a significant reduction in time to solution.

Despite the apparent simplicity of stencil computations and the fact that their computation structure maps well to current hardware architectures, meticulous architecture-specific tuning is still required to elicit a platform's full computational power due to the fact that microarchitectures have grown increasingly complex. Manual architecture-specific tuning requires a significant effort: Not only does it require a deeper understanding of the architecture, but it is also both a time consuming and error-prone process. Although the performance gain might justify the effort, the code usually becomes non-portable and hard to maintain.

The goal of PATUS is to accept an intuitive stencil specification and turn it into architecture-specific, optimized, high-performance code. The stencil specification is formulated in a small domain-specific language, which is explained in section 3.2.

Currently, PATUS supports shared-memory CPU architectures and, as a proof of concept, CUDA-programmable GPUs.

1.1 Copyrights

Cetus. A Source-to-Source Compiler Infrastructure for C Programs. Copyright under the Cetus Artistic License.
http://cetus.ecn.purdue.edu/

JGAP. Java Genetic Algorithms Package. Copyright under the GNU Lesser General Public License.
http://jgap.sourceforge.net/

Time measurement. Adapted from the time measurement in FFTW. Copyright (c) 2003, 2007-8 Matteo Frigo. Copyright (c) 2003, 2007-8 Massachusetts Institute of Technology.

2 Installation

2.1 Prerequisites

PATUS works both on Linux and on Microsoft Windows. PATUS assumes that a 64-bit operating system is installed.

There is currently no IDE integration, so the tool has to be started on the command line. However, in the `etc` directory of the PATUS distribution, you can find syntax files for `vi` and `gedit`.

Prior to using PATUS, the following software needs to be installed on your computer:

- **Java 7 or newer.** If you have an older version, please download the JRE 7 or the JDK 7 from http://www.oracle.com/technetwork/java/javase/downloads/index.html and follow the respective installation instructions.
- **Maxima.** Download from http://maxima.sourceforge.net/ and follow the respective installation instructions. For Linux distributions, installation packages are available. E.g., under Ubuntu, type
  ```
  sudo apt-get install maxima
  ```
  in a shell to install the software.
- **gcc.** To compile the code generated by PATUS, you will need a recent version of the GNU C compiler.
  On Linux, if gcc is installed the default version usually should be fine. On Windows, we recommend installing TDM-GCC, which features an easy install of the GCC toolset and MinGW-w64.
  Note that the more "sophisticated" generated code versions require a 64-bit operating system.
- **Make.** To build the generated benchmarking harness and starting the auto-tuner, PATUS generates Makefiles.
  On Linux, if build tools are installed, `make` will be available. On Windows, we recommend installing Make for Windows.

2.2 Installation

2.2.1 Linux

Unzip the `patus-0.1.zip` file in a directory of your choice.

```
unzip patus-0.1.zip
```

Change into the newly created directory.

```
cd Patus
```

Execute the shell script that will set the environment variable PATUS_HOME and modify your PATH to include the Patus directory.
In bash and sh:

```
source util/patusvars.sh
```

In csh and tcsh:

```
source util/patusvars.csh
```

These scripts assume that they are sourced from within the Patus directory.

Now you can run PATUS by typing

```
patus
```

on the command line.

2.2.2 Windows

3 Using PATUS

3.1 A Walkthrough Example

In PATUS, the user specifies the actual stencil computation. Thus, if the tool is to be used to solve differential equations using a finite difference-based discretization method, the discretization needs to be done prior to the implementation in PATUS. In the following, we show briefly how this can be done by means of a simple example.

3.1.1 From a Model to a Stencil Specification

Consider the classical wave equation on $\Omega = [-1, 1]^3$ with Dirichlet boundary conditions and some initial condition:

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - c^2 \Delta u = 0 \qquad & \text{in } \Omega, \\ u = 0 \qquad & \text{on } \partial\Omega, \\ u(x, y, z)|_{t=0} = \sin(2\pi x) \sin(2\pi y) \sin(2\pi z). & \end{aligned} \tag{1}$$

Using an explicit finite difference method to discretize the equation both in space and time by means of a fourth-order discretization of the Laplacian Δ over an equidistant spatial grid with stepsize h and a second-order scheme with time step δt in time, we obtain

$$\frac{u^{(t+\delta t)} - 2u^{(t)} + u^{(t-\delta t)}}{\delta t} - c^2 \Delta_h u^{(t)} = 0, \tag{2}$$

where Δ_h is the discretized version of the Laplacian:

$$\begin{aligned}\Delta_h u^{(t)}(x,y,z) = &-\tfrac{15}{2h^2}u^{(t)}(x,y,z)+ \\ &-\tfrac{1}{12h^2}\Big(u^{(t)}(x-2h,y,z)+u^{(t)}(x,y-2h,z)+u^{(t)}(x,y,z-2h)\Big)+ \\ &\tfrac{4}{3h^2}\Big(u^{(t)}(x-h,y,z)+u^{(t)}(x,y-h,z)+u^{(t)}(x,y,z-h)\Big)+ \\ &\tfrac{4}{3h^2}\Big(u^{(t)}(x+h,y,z)+u^{(t)}(x,y+h,z)+u^{(t)}(x,y,z+h)\Big)+ \\ &-\tfrac{1}{12h^2}\Big(u^{(t)}(x+2h,y,z)+u^{(t)}(x,y+2h,z)+u^{(t)}(x,y,z+2h)\Big).\end{aligned} \tag{3}$$

Substituting Eqn. 3 into Eqn. 2, solving Eqn. 2 for $u^{(t+\delta t)}$, and interpreting u as a grid in space and time with mesh size h and time step δt, we arrive at

$$\begin{aligned}u[x,y,z;t+1] = 2u[x,y,z;t]-u[x,y,z;t-1]+c^2\tfrac{\delta t}{h^2}\Big(&-\tfrac{15}{2}u[x,y,z;t]+ \\ &-\tfrac{1}{12}\left(u[x-2,y,z;t]+u[x,y-2,z;t]+u[x,y,z-2;t]\right)+ \\ &\tfrac{4}{3}\left(u[x-1,y,z;t]+u[x,y-1,z;t]+u[x,y,z-1;t]\right)+ \\ &\tfrac{4}{3}\left(u[x+1,y,z;t]+u[x,y+1,z;t]+u[x,y,z+1;t]\right)+ \\ &-\tfrac{1}{12}\left(u[x+2,y,z;t]+u[x,y+2,z;t]+u[x,y,z+2;t]\right)\Big).\end{aligned}$$

This can now be turned into a PATUS stencil specification almost trivially:

```
stencil wave(
  float grid U(0..x_max-1, 0..y_max-1, 0..z_max-1),
  float param fMin = -1,
  float param fDX = 2 / (x_max-3),
  float param fDT_DX_sq = 0.25)
{
  // do one timestep within the stencil kernel
  iterate while t < 1;

  // define the region on which the stencil is evaluated
  domainsize = (2..x_max-3, 2..y_max-3, 2..z_max-3);

  // define the initial condition (how the data is initialized
  // before the computation)
  // note that 0 <= x < x_max, etc.
  initial {
    U[x,y,z; -1] = sinf(2*π*((x-1)*fDX+fMin)) *
      sinf(2*π*((y-1)*fDX+fMin)) * sinf(2*π*((z-1)*fDX+fMin));
    U[x, y, z; -1 :
      x==0 || y==0 || z==0 || x==x_max-1 || y==y_max-1 || z==z_max-1 ] = 0;
    U[x,y,z; 0] = U[x,y,z; -1];
    U[x,y,z; 1] = 0;
  }

  // define the actual stencil computation
  operation {
    float c1 = 2 - 15/2 * fDT_DX_sq;
    float c2 = 4/3 * fDT_DX_sq;
    float c3 = -1/12 * fDT_DX_sq;

    U[x,y,z; t+1] = c1 * U[x,y,z; t] - U[x,y,z; t-1] +
      c2 * (
        U[x+1,y,z; t] + U[x-1,y,z; t] +
        U[x,y+1,z; t] + U[x,y-1,z; t] +
        U[x,y,z+1; t] + U[x,y,z-1; t]
      ) +
      c3 * (
        U[x+2,y,z; t] + U[x-2,y,z; t] +
        U[x,y+2,z; t] + U[x,y-2,z; t] +
```

```
        U[x,y,z+2; t] + U[x,y,z-2; t]
      );
  }
}
```

From this stencil specification, PATUS will generate

- a C source code file implementing the stencil computation and
- source files from which a benchmarking harness can be built.

The benchmarking executable is then used by the auto-tuner, which determines a set of architecture-specific parameters, for which the stencil achieves the best performance.

3.1.2 The Specification Explained

- The **stencil** specification defines a stencil "wave," which operates on a grid (called "U") of size [0, x_max-1] $\times$ [0, y_max-1] $\times$ [0,z_max-1]. Note that the size parameters do not have to be defined in the stencil specification. Instead, they will appear as arguments to the generated stencil kernel function, and, in the benchmarking harness, as command line arguments.

  The remaining arguments, "fMin," "fDX," and "fDT_DX_sq," are used for initializing the grid and performing the stencil computation. These parameters will also appear as arguments to the generated C function implementing the computation.

  Optionally, parameters can be initialized with default values (as shown in the listing above). This affects only the benchmarking harness and will fix the values which are passed to the generated stencil kernel.

- The **iterate while** statement defines the number of timesteps to be performed within one stencil kernel call. In the example, one timestep per kernel invocation will be performed. (Timesteps are counted from 0.) The statement can be omitted. Then, the number of timesteps defaults to 1. In the future we will also allow (reduction-based) convergence criteria (e.g., "iterate as long as the residual is larger than some ε").

- The **domainsize** defines the iteration space. While the total size of the grid extends from 0 to $*$_max-1, the stencil is only applied to the points between 2 and $*$_max-3 ($* \in \{x, y, z\}$), i.e., only to the *interior* grid points.

- The **initial** block defines how the grid is initialized, or more mathematically: it defines the initial condition of the (discretized) PDE. The "sinf" function is actually a C function (single precision sine) which will be called (this is the behavior if there is a function which is not known to PATUS). Note that PATUS also defines the π literal with the obvious meaning.
  The second statement of the initialization sets all the grid points for which x, y, z are 0 or $*$_max-1 (the condition after colon) to zero. I.e., you can use the set builder notation (with any logical and comparison operators you know from C/C++) to select certain grid points and initialize them.

initial blocks are not mandatory; if no **initial** is provided, PATUS will create an initialization routine anyway, initializing the data with arbitrary values. (This is to ensure the correct data placement on NUMA machines.)

- The **operation**, finally, defines the actual stencil computation. It can contain definitions of constants (as in the listing) or temporary values, and it can also contain more than one stencil expression.
- Optionally, the stencil specification can also contain a **boundaries** block, in which special treatment of boundary regions can be specified. Essentially, within the **boundaries** block, special stencils are defined which are applied to boundary regions. See section 3.2.

3.1.3 Building the Benchmarking Harness

Once the stencil specification is written, PATUS can be run to transform it into C code. Assume the stencil specification was saved in the file `examples/stencils/wave-1.stc`. (This very stencil specification is actually there.) In your shell, type

```
cd examples/stencils
patus wave-1.stc
```

This will generate the C code implementing the Wave stencil, using the default architecture, and putting the generated files in the `out` directory. (You can change the output directly by adding the `--outdir=<your-output-dir>` to the PATUS command line.) Change into the output directory,

```
cd out
```

and type

```
make
```

to build the benchmarking harness, which will be used for the auto-tuning process, and can also be used for simple simulations. When the build completes successfully, there is an executable file, `bench` (or `bench.exe` on Windows) in the directory.

If you try to start the benchmarking executable, you will see that it expects some command line arguments:

```
$ ./bench
Wrong number of parameters.  Syntax:
./bench <x_max> <y_max> <z_max> <cb_x> <cb_y> <cb_z> <chunk> <_unroll_p3>
```

The $*$_max correspond to the unspecified domain size variables in the stencil specification. All the other arguments come from the code generator, and it is the auto-tuner's task to find the best values for them.

3.1.4 Auto-Tuning

The Makefile (which was also used to build the benchmarking harness) defines a "tune" target, which starts the auto-tuner. "tune" expects the domain size variables to be specified:

```
make tune x_max=64 y_max=64 z_max=64
```

Type the above command in the shell. The auto-tuner will run the bench-

marking executable a number of times, varying the values of the arguments to determine, and it will terminate with a message like

```
Optimal parameter configuration found:
64 64 64 62 20 8 2 0

Timing information for the optimal run:
0.9047935972598529

Program output of the optimal run:
Flops / stencil call:  16
Stencil computations:  1116000
Bytes transferred:     15728640
Total Flops:           17856000
Seconds elapsed:       0.000423
Performance:           42.193517 GFlop/s
Bandwidth utilization: 37.166590 GB/s
1352692.000000
Validation OK.
```

Thus, according to the auto-tuner, on the machine the benchmark was run and for x_max=y_max=z_max=64, the best parameter combination is

```
cb_x=62, cb_y=20, cb_z=8, chunk=2, _unroll_p3=0.
```

3.1.5 Simple Visualization

You can run the benchmarking executable with the arguments provided by the auto-tuner,

```
./bench 64 64 64 62 20 8 2 0
```

If you add the flag `-o` to the command line,

```
./bench 64 64 64 62 20 8 2 0 -o
```

the program will take snapshots of the data grids before and after the execution of the stencil kernel and write them to text files. If you have gnuplot installed on your system, you can run

```
make plot
```

which will use gnuplot to create image files from the data text files.

3.2 Writing Your Own Stencil Specifications

3.2.1 Stencil Arguments

3.2.2 The “operation”

stencil offsets have to be compile-time constants

```
stencil wave(
  float grid U(0..x_max-1, 0..y_max-1, 0..z_max-1),
  float param fDT_DX_sq = 0.25)
{
  // define the region on which the stencil is evaluated
  domainsize = (2..x_max-3, 2..y_max-3, 2..z_max-3);

  // define the actual stencil computation
  operation {
    float c[0..2] = {
      2 - 15/2 * fDT_DX_sq,
```

```
      4/3 * fDT_DX_sq,
      -1/12 * fDT_DX_sq
    };

    U[x,y,z; t+1] = c[0]*U[x,y,z; t] - U[x,y,z; t-1] +
      { i=-1..1, j=-1..1, k=-1..1, r=1..2 : i^2+j^2+k^2==1 } sum(
        c[r] * U[x+r*i,y+r*j,z+r*k; t]
      );
  }
}
```

3.2.3 Number of Timesteps and Iteration Space

3.2.4 Boundary Conditions

3.2.5 Initial Condition

set comprehensions/set-builder notation

3.3 Integrating Into Your Own Application

Two modes.

See the directory `examples/applications` for examples how to integrate PATUS-generated code into user code.

Create Fortran example

4 Current Limitations

In the current release, there are the following limitations to PATUS:

- Only shared memory architectures are supported directly (specifically: shared memory CPU systems and single-GPU setups). However, if you have MPI code that handles communication and synchronization for the distributed case, PATUS-generated code can be used as a replacement for the per-node stencil computation code.
- It is assumed that the evaluation order of the stencils within one spatial sweep is irrelevant. Also, always all points within the domain are traversed per sweep. One grid array is read and another array is written to (Jacobi iteration). In particular, this rules out schemes with special traversal rules such as red-black Gauss-Seidel iterations.
- Boundary handling is not yet optimized in the generated code. Expect performance drops if a boundary specification is included in the stencil specification.
- There is no support yet for temporally blocked schemes.
- Limitation for Fortran interoperability: only one timestep per stencil kernel is supported.
- Limitation for (CUDA-capable) GPUs: the generated code does not do any significant performance optimizations yet. Also, the stencil kernel can only perform one timestep (since CUDA does not allow global synchronization from within a kernel).

5 PATUS Grammars

5.1 Stencil DSL Grammar

In the following, the EBNF grammar for the PATUS stencil specifications syntax is given. The grayed out identifiers have not yet been specified or implemented and will be added eventually in the future.

⟨*Stencil*⟩ ::= `stencil' ⟨*Identifier*⟩ `{' (⟨*Options*⟩) ⟨*DomainSize*⟩ ⟨*NumIterations*⟩ ⟨*Operation*⟩ ⟨*Boundary*⟩ (⟨*Filter*⟩) (⟨*StoppingCriterion*⟩) `}'

⟨*DomainSize*⟩ ::= `domainsize' `=' ⟨*Box*⟩ `;'

⟨*NumIterations*⟩ ::= `t_max' `=' ⟨*IntegerExpr*⟩ `;'

⟨*Operation*⟩ ::= `operation' ⟨*Identifier*⟩ `(' ⟨*ParamList*⟩ `)' `{' { ⟨*Statement*⟩ } `}'

⟨*ParamList*⟩ ::= { ⟨*GridDecl*⟩ | ⟨*ParamDecl*⟩ }

⟨*Statement*⟩ ::= ⟨*LHS*⟩ `=' ⟨*StencilExpr*⟩ `;'

⟨*LHS*⟩ ::= ⟨*StencilNode*⟩ | ⟨*VarDecl*⟩

⟨*StencilExpr*⟩ ::= ⟨*StencilNode*⟩ | ⟨*Identifier*⟩ | ⟨*NumberLiteral*⟩ | ⟨*FunctionCall*⟩ | (⟨*UnaryOperator*⟩ ⟨*StencilExpr*⟩) | (⟨*StencilExpr*⟩ ⟨*BinaryOperator*⟩ ⟨*StencilExpr*⟩) | `(' ⟨*StencilExpr*⟩ `)'

⟨*StencilNode*⟩ ::= ⟨*Identifier*⟩ `[' ⟨*SpatialCoords*⟩ ( `;' ⟨*TemporalCoord*⟩) (`;' ⟨*ArrayIndices*⟩ ) `]'

⟨*SpatialCoords*⟩ ::= (`x' | `y' | `z' | `u' | `v' | `w' | `x' ⟨*IntegerLiteral*⟩) (⟨*Offset*⟩)

⟨*TemporalCoord*⟩ ::= `t' (⟨*Offset*⟩

⟨*ArrayIndices*⟩ ::= ⟨*IntegerLiteral*⟩ { `,' ⟨*IntegerLiteral*⟩ }

⟨*Offset*⟩ ::= ⟨*UnaryOperator*⟩ ⟨*IntegerLiteral*⟩

⟨*FunctionCall*⟩ ::= ⟨*Identifier*⟩ `(' ( ⟨*StencilExpr*⟩ { `,' ⟨*StencilExpr*⟩ }) `)'

⟨*IntegerExpr*⟩ ::= ⟨*Identifier*⟩ | ⟨*IntegerLiteral*⟩ | ⟨*FunctionCall*⟩ | (⟨*UnaryOperator*⟩ ⟨*IntegerExpr*⟩) | (⟨*IntegerExpr*⟩ ⟨*BinaryOperator*⟩ ⟨*IntegerExpr*⟩) | `(' ⟨*IntegerExpr*⟩ `)'

⟨*VarDecl*⟩ ::= ⟨*Type*⟩ ⟨*Identifier*⟩

⟨*Box*⟩ ::= `(' ⟨*Range*⟩ { `,' ⟨*Range*⟩ `)'

⟨*Range*⟩ ::= ⟨*IntegerExpr*⟩ `..' ⟨*IntegerExpr*⟩

⟨*GridDecl*⟩ ::= (⟨*Specifier*⟩) ⟨*Type*⟩ `grid' ⟨*Identifier*⟩ ( `(' ⟨*Box*⟩ `)') (⟨*ArrayDecl*⟩)

⟨*ParamDecl*⟩ ::= ⟨*Type*⟩ `param' ⟨*Identifier*⟩ (⟨*ArrayDecl*⟩)

⟨*ArrayDecl*⟩ ::= `(' ⟨*IntegerLiteral*⟩ { `,' ⟨*IntegerLiteral*⟩ } `)'

⟨*Specifier*⟩ ::= `const'

⟨*Type*⟩ ::= `float' | `double'

⟨*UnaryOperator*⟩ ::= `+' | `-'

⟨*BinaryOperator*⟩ ::= `+' | `-' | `*' | `/' | `^'

5.2 Strategy DSL Grammar

The following EBNF grammar specifies the PATUS Strategy syntax. Again, as the project matures, the specification might change so that yet missing aspects of parallelization and optimization methods can be specified as PATUS Strategies.

⟨*Strategy*⟩ ::= `strategy' ⟨*Identifier*⟩ `(' ⟨*ParamList*⟩ `)' ⟨*CompoundStatement*⟩

⟨*ParamList*⟩ ::= ⟨*SubdomainParam*⟩ { `,' ⟨*AutoTunerParam*⟩ }

⟨*SubdomainParam*⟩ ::= `domain' ⟨*Identifier*⟩

⟨*AutoTunerParam*⟩ ::= `auto' ⟨*AutoTunerDeclSpec*⟩ ⟨*Identifier*⟩

⟨*AutoTunerDeclSpec*⟩ ::= `int' | `dim' | (`codim' `(' ⟨*IntegerLiteral*⟩ `)')

⟨*Statement*⟩ ::= ⟨*DeclarationStatement*⟩ | ⟨*AssignmentStatement*⟩ | ⟨*CompoundStatement*⟩ | ⟨*IfStatement*⟩ | ⟨*Loop*⟩

⟨*DeclarationStatement*⟩ ::= ⟨*DeclSpec*⟩ ⟨*Identifier*⟩ `;'

⟨*AssignmentStatement*⟩ ::= ⟨*LValue*⟩ `=' ⟨*Expr*⟩ `;'

⟨*CompoundStatement*⟩ ::= `{' { ⟨*Statement*⟩ } `}'

⟨*IfStatement*⟩ ::= `if' `(' ⟨*ConditionExpr*⟩ `)' ⟨*Statement*⟩ ( `else' ⟨*Statement*⟩)

⟨*Loop*⟩ ::= (⟨*RangeIterator*⟩ | ⟨*SubdomainIterator*⟩) (`parallel' ( ⟨*IntegerLiteral*⟩ ) ( `schedule' ⟨*IntegerLiteral*⟩)) ⟨*Statement*⟩

⟨*RangeIterator*⟩ ::= `for' ⟨*Identifier*⟩ `=' ⟨*Expr*⟩ `..' ⟨*Expr*⟩ ( `by' ⟨*Expr*⟩)

⟨*SubdomainIterator*⟩ ::= `for' ⟨*SubdomainIteratorDecl*⟩ `in' ⟨*Identifier*⟩ `(' ⟨*Range*⟩ `;' ⟨*Expr*⟩ `)'

⟨*SubdomainIteratorDecl*⟩ ::= ⟨*PointDecl*⟩ | ⟨*PlaneDecl*⟩ | ⟨*SubdomainDecl*⟩

⟨*PointDecl*⟩ ::= `point' ⟨*Identifier*⟩

⟨*PlaneDecl*⟩ ::= `plane' ⟨*Identifier*⟩

⟨*SubdomainDecl*⟩ ::= `subdomain' ⟨*Identifier*⟩ `(' ⟨*Range*⟩ `)'

⟨*Range*⟩ ::= ⟨*Vector*⟩ { ⟨*UnaryOperator*⟩ ⟨*ScaledBorder*⟩ }

⟨*Vector*⟩ ::= ⟨*Subvector*⟩ (`...' ( `,' ⟨*Subvector*⟩))

⟨*Subvector*⟩ ::= (`:' { `,' ⟨*ScalarList*⟩ }) | ⟨*DimensionIdentifier*⟩ | ⟨*DomainSizeExpr*⟩ | ⟨*BracketedVector*⟩ | ⟨*ScalarList*⟩

⟨*ScalarList*⟩ ::= ⟨*ScalarRange*⟩ { `,' ⟨*ScalarRange*⟩ }

⟨*DimensionIdentifier*⟩ ::= ⟨*Expr*⟩ (`(' ⟨*Vector*⟩ `)')

⟨*DomainSizeExpr*⟩ ::= ⟨*SizeProperty*⟩ (`(' ⟨*Vector*⟩ `)')

⟨*BracketedVector*⟩ ::= `(' ⟨*Vector*⟩ { `,' ⟨*Vector*⟩ } `)'

⟨*ScalarRange*⟩ ::= ⟨*Expr*⟩ (`..' ⟨*Expr*⟩)

⟨*SizeProperty*⟩ ::= (`stencil' | ⟨*Identifier*⟩ ) `.' (`size' | `min' | `max')

⟨*ScaledBorder*⟩ ::= (⟨*Expr*⟩ ⟨*MultiplicativeOperator*⟩) ⟨*Border*⟩ (⟨*MultiplicativeOperator*⟩ ⟨*Expr*⟩)

⟨*Border*⟩ ::= ⟨*StencilBoxBorder*⟩ | ⟨*LiteralBorder*⟩

⟨*StencilBoxBorder*⟩ ::= `stencil' `.' `box' ( `(' ⟨*Vector*⟩ `)')

⟨*LiteralBorder*⟩ ::= `(' ⟨*Vector*⟩ `)' `,' `(' ⟨*Vector*⟩ `)'

⟨*LValue*⟩ ::= ⟨*GridAccess*⟩ | ⟨*Identifier*⟩

⟨*GridAccess*⟩ ::= ⟨*Identifier*⟩ `(' ⟨*SpatialIndex*⟩ `;' ⟨*Expr*⟩ { `;' ⟨*Expr*⟩ } `)'

⟨*SpatialIndex*⟩ ::= ⟨*Identifier*⟩ | ⟨*Range*⟩

⟨*Expr*⟩ ::= ⟨*Identifier*⟩ | ⟨*NumberLiteral*⟩ | ⟨*FunctionCall*⟩ | (⟨*UnaryOperator*⟩ ⟨*Expr*⟩) | (⟨*Expr*⟩ ⟨*BinaryOperator*⟩ ⟨*Expr*⟩) | `(' ⟨*Expr*⟩ `)'

⟨*FunctionCall*⟩ ::= ⟨*Identifier*⟩ `(' ( ⟨*Expr*⟩ { `,' ⟨*Expr*⟩ }) `)'

⟨*ConditionExpr*⟩ ::= ⟨*ComparisonExpr*⟩ | (⟨*ConditionExpr*⟩ ⟨*LogicalOperator*⟩ ⟨*ConditionExpr*⟩)

⟨*ComparisonExpr*⟩ ::= ⟨*Expr*⟩ ⟨*ComparisonOperator*⟩ ⟨*Expr*⟩

⟨*UnaryOperator*⟩ ::= `+' | `-'

⟨*MultiplicativeOperator*⟩ ::= `*'

⟨*BinaryOperator*⟩ ::= `+' | `-' | `*' | `/' `%'

⟨*LogicalOperator*⟩ ::= `||' | `&&'

⟨*ComparisonOperator*⟩ ::= `<' | `<=' | `==' | `>=' | `>' | `!='

References