Patus Documentation

Matthias-M. Christen
m.christen@unibas.ch
University of Basel, Switzerland

Contents

Copyrights

Chapter 1

A Patus Walkthrough Example

In this section, we give an example, who a PATUS stencil specification can be derived from a mathematical problem description. From this stencil specification, the PATUS code generator generates a C code which implements the compute kernel along with a benchmarking harness that can be used to measure the performance of the generated code and also shows how the kernel is to be called from within a user code.

1.1 From a Model to a Stencil

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

$$\begin{aligned} \frac{\partial^2 u}{\partial t^2} - c^2 \Delta u &= 0 \qquad \text{in } \Omega, \\ u &= g \qquad \text{on } \partial\Omega, \\ u|_{t=0} &= f. \end{aligned} \tag{1.1}$$

We use an explicit finite difference method to discretize the equation both in space and time. For the discretization in time we use a second-order scheme with time step δt. For the discretization in space we choose a fourth-order discretization of the Laplacian Δ on the structured uniformly discretized grid Ω_h with step size h. This discretization gives us

$$\frac{u^{(t+\delta t)} - 2u^{(t)} + u^{(t-\delta t)}}{\delta t} - c^2 \Delta_h u^{(t)} = 0, \tag{1.2}$$

where Δ_h is the discretized version of the Laplacian:

$$\begin{aligned}\Delta_h u^{(t)}(x,y,z) = {} & \tfrac{-15}{2h^2} u^{(t)}(x,y,z) + \\ & \tfrac{-1}{12h^2}\left(u^{(t)}(x-2h,y,z) + u^{(t)}(x,y-2h,z) + u^{(t)}(x,y,z-2h)\right) + \\ & \tfrac{4}{3h^2}\left(u^{(t)}(x-h,y,z) + u^{(t)}(x,y-h,z) + u^{(t)}(x,y,z-h)\right) + \\ & \tfrac{4}{3h^2}\left(u^{(t)}(x+h,y,z) + u^{(t)}(x,y+h,z) + u^{(t)}(x,y,z+h)\right) + \\ & \tfrac{-1}{12h^2}\left(u^{(t)}(x+2h,y,z) + u^{(t)}(x,y+2h,z) + u^{(t)}(x,y,z+2h)\right).\end{aligned} \tag{1.3}$$

Substituting Eqn. 1.3 into Eqn. 1.2, solving Eqn. 1.2 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

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

To actually solve the discretized equation, we need to specify the mesh size h or, equivalently, the number of grid points N, and the time step δt or, equivalently, the number of time steps $t_{\max}$. Choosing a concrete number of the number of time steps, we can transform the above equation together with the number of grid points and the number of time steps almost trivially into a PATUS stencil specification:

Example 1.1: *A* PATUS *stencil specification.*

The listing below shows the PATUS stencil specification for the classical wave equation discretized by 4^{th} order finite differences in space and by 2^{nd} order finite differences in time. Note that the maximum N in the domain size specification is inclusive.

```
stencil wave
{
    domainsize = (1 .. N, 1 .. N, 1 .. N);
    t_max = 100;

    operation (float grid u, float param c2dt_h2)
    {
        u[x, y, z; t+1] = 2 * u[x, y, z; t] - u[x, y, z; t-1] +
            c2dt_h2 * (
                -15/2 * u[x, y, z; t] +
                4/3 * (
```

Example 1.1: *A* PATUS *stencil specification.* (cont.)

```
                    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]
                )
                -1/12 * (
                    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]
                )
            );
    }
}
```

1.2 Generating The Code

We feed this stencil specification as an input to PATUS, which will turn it to C code. PATUS expects two other input files: a template defining how the code will be parallelized and how code optimizations will be applied, e.g., how loop tiling/cache blocking is applied. In PATUS lingo, this is called a *Strategy*. The other input is a description of the hardware architecture. It defines which code generation back-end to use (e.g., the OpenMP paradigm for shared memory CPU system, or NVIDIA C for CUDA for NVIDIA GPUs), and how arithmetic operations and data types are mapped to the corresponding vector intrinsics and vector data types, for instance.

Example 1.2: *Generating the C code from a stencil specification.*

```
java -jar patus.jar codegen --stencil=wave.stc
    --strategy=cacheblocking.stg
    --architecture="arch/architectures.xml,Intel x86_64 SSE"
    --outdir=output
```

The command in Ex. 1.2 will create an implementation for the stencil kernel specified in the file `wave.stc` and a benchmarking harness for that kernel and file it in the directory `output`. The Strategy chosen is a cache blocking strategy defined in the file `cacheblocking.stg`, which comes with the PATUS software. The hardware architecture for which the code will be generated is defined by the identifier `Intel x86_64 SSE`, which can be found in the architecture definition file, `arch/architectures.xml`.

After running the PATUS code generation, the directory `output` will contain the following files:

- `kernel.c` — The implementation of the stencil kernel defined in `wave.stc`.

- `driver.c` — The benchmarking harness invoking the stencil kernel and measuring the time for the stencil call. It allocates and initializes data with arbitrary values and (by default) validates the result returned by the stencil kernel by comparing it to a naïve sequential implementation.

- `timer.c` — Functions related to timing and calculating the performance.

- `patusrt.h` — Header file for the timing functions.

- `cycle.h` — Functions for counting clock cycles to do the timing. (This code has been taken from FFTW [1].)

- `Makefile` — A GNUmake Makefile to build the benchmarking harness.

The benchmarking harness then can be built by typing `make` on the command line. This will compile and link the generated code and produce the benchmarking executable, by default called `bench`.

1.3 Running and Tuning

The benchmark executable requires a number of parameters to run:

Example 1.3: *Starting the benchmark executable.*

```
chrmat@palu1:wave> ./bench
Wrong number of parameters.  Syntax:
./bench <N> <cb_x> <cb_y> <cb_z> <chunk> <_unroll_p3>
```

`N` corresponds to the N used in the stencil specification for the definition of the domain size. If additional identifiers would have been used to define the domain size, they would appear as parameters to the executable as well. The `cb_x`, `cb_y`, `cb_z`, and `chunk` arguments come from the Strategy. They specify the sizes of the cache blocks in x, y, and z directions and the number of consecutive cache blocks assigned to one thread. PATUS unrolls the inner-most loop nest containing the stencil evaluation. `_unroll_p3` selects one of the unroll configuration code variants: by default, PATUS creates code variants with the loop nest unrolled once (i.e., no unrolling is done) or twice in each direction. Since the example wave stencil is defined in 3 dimensions, there are $2^3 = 8$ code variants with different unrolling configurations.

In Ex. 1.4, the benchmark executable was run with a domain size of 200^3 grid points and an arbitrary cache block size of $16 \times 16 \times 16$ grid points per block, one block in a packet per thread, and the 0^{th} loop unrolling configuration.

Example 1.4: *Running the benchmark executable with arbitrary parameters.*

```
chrmat@palu1:wave> ./bench 200 16 16 16 1 0
Flops / stencil call:  19
Stencil computations:  40000000
Bytes transferred:     509379840
Total Flops:           760000000
Seconds elapsed:       0.230204
Performance:           3.301418 GFlop/s
Bandwidth utilization: 2.212731 GB/s
506450156.000000
Validation OK.
```

The benchmark executable prints the number of floating point operations per stencil evaluation, the total number of stencil evaluations that were performed for the time measurement, the number of transferred bytes and the total number of floating point operations, the time it took to complete the sweeps and the calculated performance in GFlop/s and bandwidth utilization. The number below that is a representation of the time spent in the compute kernel, on which the auto-tuner bases the search. (It tries to minimize that number.) The string "Validation OK" says that the validation test (against the naïve, sequential implementation) was passed, i.e., the relative errors did not exceed certain bounds.

The idea is that the user chooses the problem size, N, but all the other parameters, which do not change the problem definition, but rather implementation details which affect the performance, are to be chosen by the auto-tuner. In the current state of the software, the user still needs some knowledge how to find the performance-specific parameters (in this case `cb_x`, `cb_y`, `cb_z`, `chunk`, `_unroll_p3`). An idea for future work is to encapsulate this knowledge in the Strategy, which defines the parameters, so that an auto-tuner configuration script can be generated in order to fully automate the auto-tuning process.

We choose $N = 200$. Experience tells us that in cases with relatively small domain size (such as $N = 200$), a good choice for `cb_x` is N. There are several reasons why cutting the domain in the x direction, which is the unit stride direction (i.e. choosing `cb_x` $< N$), is a bad idea. Reading data in a streaming fashion from DRAM is faster than jumping between addresses. Utilizing full cache lines maximizes data locality. And the hardware prefetcher is most effective when the constant-stride data streams are as long as possible. In Ex. 1.5 we let the auto-tuner choose `cb_y`, `cb_z`, `chunk`, and `_unroll_p3` by letting `cb_y` and `cb_z` from 4 to $N = 200$ in increments of 4 (`4:4:200`), and we want the auto-tuner to try all the powers of 2 between 1 and 16 (`1:*2:16!`) for the fifth command line argument, `chunk`, and try all the values between 0 and 7 (`0:7!`) for `_unroll_p3`, by which all the generated loop unrolling variants are referenced. The exclamation mark specifies that the corresponding parameter is searched exhaustively, i.e., the auto-tuner is instructed to visit all of the values in the range specified.

Example 1.5: *Running the* PATUS *auto-tuner.*

```
java -jar patus.jar autotune ./bench 200 200 4:4:200 4:4:200
   1:*2:16!  0:7!  "C((\$1+\$3-1)/\$3)*((\$1+\$4-1)/\$4)>=$OMP_NUM_THREADS"
```

The auto-tuner also allows to add constraints, such as the expression in the last argument in the call in Ex. 1.5. Constraints are preceded by a `C` and can be followed by any comparison expression involving arithmetic. Parameter values are referenced by a number preceded by a dollar sign \$; the numbering starts with 1. In Ex. 1.5, the condition reads $(\$1 + \$3 - 1)/\$3 \cdot (\$1 + \$4 - 1)/\$4 \geqslant T$. The sub-expression $(\$1 + \$3 - 1)/\$3$ is the number of blocks in the y-direction, using the integer division (which rounds towards zero) to express the missing ceiling function*). Similarly, $(\$1 + \$4 - 1)/\$4$ is the number of blocks in z-direction. Thus, the condition states that the total number of blocks must be greater or equal than the number of threads executing the program (assuming the environment variable `$OMP_NUM_THREADS` is set and controls how many OpenMP threads the program uses. Adding constraints is optional, but they can reduce the number of searches by restricting the search space. In this case, we simply exclude the points in the search space with bad performance, but constraints can be a helpful tool to suppress invalid configurations. For instance, the numbers of threads per block on a GPU must not exceed a particular number lest the program fails. (The numbers are 512 threads per block on older graphics cards, and 1024 threads per block on Fermi GPUs).

Ex. 1.6 shows an excerpt of the auto-tuner output. The program is run for many parameter configurations, and at the end of the search, the auto-tuner displays the parameter configuration and output of the run for which the best performance was achieved.

Example 1.6: *Output of the auto-tuner.*

```
./bench 200 200 4:4:200 4:4:200 1:*2:16!  0:7!
Parameter set { 200 }
Parameter set { 200 }
Parameter set { 4, 8, ...}
Parameter set { 4, 8, ...}
Parameter set , Exhaustive { 1, 2, 4, 8, 16 }
Parameter set , Exhaustive { 0, 1, 2, 3, 4, 5, 6, 7 }
Using optimizer:  Powell search method
Executing [./bench, 200, 200, 4, 4, 1, 0]...
...

200 200 36 160 1 3
1.5052755E8
```

*For positive integers a, b and the integer division $\div$ (defined by $a \div b := \left\lfloor \frac{a}{b} \right\rfloor$, it holds $\left\lceil \frac{a}{b} \right\rceil = (a + b - 1) \div b$.

Example 1.6: *Output of the auto-tuner.* (cont.)

```
Program output of the optimal run:
Flops / stencil call:  19
Stencil computations:  40000000
Bytes transferred:     509379840
Total Flops:           760000000
Seconds elapsed:       0.068421
Performance:           11.107680 GFlop/s
Bandwidth utilization: 7.444774 GB/s
150527550.000000
Validation OK.
```

Chapter 2

Patus Usage

2.1 Code Generation

The PATUS code generator is invoked by the following command:

```
java -jar patus.jar codegen
  --stencil=<Stencil File>  --strategy=<Strategy File>
  --architecture=<Architecture Description File>,<Hardware Name>
  [--outdir=<Output Directory>] [--generate=<Target>]
  [--kernel-file=<Kernel Output File Name>]
  [--compatibility={ C | Fortran }]
  [--unroll=<UnrollFactor1>,<UnrollFactor2>,...]
  [--use-native-simd-datatypes={ yes | no }]
  [--create-validation={ yes | no }]
  [--validation-tolerance=<Tolerance>]
  [--debug=<DebugOption1>,[<DebugOption2>,[...,[<DebugOptionN>]...]]
```

`--stencil=`*<Stencil File>*

Specifies the stencil specification file for which to generate code.

`--strategy=`*<Strategy File>*

The Strategy file describing the parallelization/optimization strategy.

`--architecture=`*<Architecture Description File>*,*<Hardware Name>*

The architecture description file and the name of the selected architecture (as specified in the `name` attribute of the `architectureType` element).

`--outdir=`<*Output Directory*>

The output directory into which the generated files will be written. Optional; if not specified the generated files will be created in the current directory.

`--generate=`<*Target*>

The target that will be generated. <*Target*> can be one of:

- `benchmark`
  This will generate a full benchmark harness. This is the default setting.
- `kernel`
  This will only generate the kernel file.

`--kernel-file=`<*Kernel Output File Name*>

Specifies the name of the C source file to which the generated kernel is written. The suffix is appended or replaced from the definition in the hardware architecture description.
Defaults to `kernel.c`.

`--compatibility={C | Fortran}`

Selects whether the generated code has to be compatible with Fortran (Omits the double pointer output type in the kernel declaration; therefore multiple time steps are not supported.) Defaults to `C`.

`--unroll=`<*UnrollFactor1*>,<*UnrollFactor2*>,...

A list of unrolling factors applied to the inner most loop nest containing the stencil computation. The unrolling factors are applied to all the dimensions.

`--use-native-simd-datatypes={yes | no}`

Specifies whether the native SSE data type is to be used in the kernel signature. If set to `yes`, this also requires that the fields are padded correctly in unit stride direction.
Defaults to `no`.

`--create-validation={yes | no}`

Specifies whether to create code that will validate the result. If <*Target*> is not `benchmark`, this option will be ignored.
Defaults to `yes`.

`--validation-tolerance=`<*Tolerance*>

Sets the tolerance for the relative error in the validation. This option is only relevant if validation code is generated (`--create-validation=yes`). Defaults to `yes`.

`--debug=`<*DebugOption1*>,[<*DebugOption2*>,[...,[<*DebugOptionN*>]...]]

Specifies debug options (as a comma-separated list) that will influence the code generator. Valid debug options (for <*DebugOptioni*>, $i = 1, \ldots, N$) are:

- `print-stencil-indices`
  This will insert a `printf` statement for every stencil calculation with the index into the grid array at which the result is written.
- `print-validation-errors`
  Prints all values if the validation fails. The option is ignored if no validation code is generated.

2.2 Auto-Tuning

The PATUS auto-tuner is invoked on the command line like so:

```
java -jar patus.jar autotune
  <Executable Filename>
  <Param1 > <Param2 > ... <ParamN >
  [ <Constraint1> <Constraint2> ... <ConstraintM> ]
  [ -m<Method> ]
```

<*Executable Filename*> is the path to the file name of the benchmark executable. The benchmark executable must expose the tunable parameters as command line parameters. The PATUS auto-tuner only generates numerical parameter values. If the benchmark executable requires strings, these must be mapped from numerical values internally in the benchmarking program.

The parameters *ParamI*, $I = 1, \ldots, N$, define integer parameter ranges and have the following syntax:

<*StartValue*>:[[*]<*Step*>:]<*EndValue*>[!]

or

<*Value1*>[,<*Value2*>[,<*Value3*>...]][!]

The first version, when no asterisk * in the <*Step*> is specified, enumerates all values in

$$\{a := \text{<}StartValue\text{>} + k \cdot \text{<}Step\text{>} : k \in \mathbb{N}_0 \text{ and } a \leqslant \text{<}EndValue\text{>}\}.$$

If no <*Step*> is given, it defaults to 1. If there is an asterisk * in front of the <*Step*>, the values enumerated are

$$\{a := \text{<}StartValue\text{>} \cdot \text{<}Step\text{>}^k : k \in \mathbb{N}_0 \text{ and } a \leqslant \text{<}EndValue\text{>}\}.$$

In the second version, all the comma-separated values <*Value1*>, <*Value2*>, ... are enumerated.

If the optional `!` is appended to the value range specification, each of the specified values is guaranteed to be used, i.e., an exhaustive search is used for the corresponding parameter.

Example 2.1: *Specifying parameter ranges.*

`1 : 10`
enumerates all the integer numbers between and including 1 and 10.

`2 : 2 : 41`
enumerates the integers $2, 4, 6, \ldots, 38, 40$.

`1 : *2 : 128`
enumerates some powers of 2, namely $1, 2, 4, 8, 16, 32, 64, 128$.

Optional constraints can be specified to restrict the parameter values. The syntax for the constraints is

`C`<*ComparisonExpression*>

where <*ComparisonExpression*> is an expression which can contain arithmetic operators `+`, `-`, `*`, and `/`, as well as comparison operators `<`, `<=`, `==`, `>=`, `>`, `!=`, and variables `$1`, ..., `$`N, which correspond to the parameters <*Param1* >, ... <*ParamN* >.

Example 2.2: *Constraints examples.*

`C$2 <= $1`
forces the second parameter to be less or equal to the first.

`C($2 + $1 - 1)/$1 >= 24`
forces $\frac{\$2+\$1-1}{\$1} = \left\lceil \frac{\$2}{\$1} \right\rceil \geqslant 24$.

The PATUS auto-tuner supports a range of search methods. The method can be selected by `-m`<*Method*> where <*Method*> is one of

- `ch.unibas.cs.hpwc.patus.autotuner.DiRectOptimizer`
  DIRECT method
- `ch.unibas.cs.hpwc.patus.autotuner.ExhaustiveSearchOptimizer`
  exhaustive search
- `ch.unibas.cs.hpwc.patus.autotuner.GeneralCombinedEliminationOptimizer`
  general combined elimination
- `ch.unibas.cs.hpwc.patus.autotuner.GreedyOptimizer`
  greedy search
- `ch.unibas.cs.hpwc.patus.autotuner.HookeJeevesOptimizer`
  Hooke-Jeeves algorithm
- `ch.unibas.cs.hpwc.patus.autotuner.MetaHeuristicOptimizer`
  genetic algorithm
- `ch.unibas.cs.hpwc.patus.autotuner.RandomSearchOptimizer`
  draws 500 random samples
- `ch.unibas.cs.hpwc.patus.autotuner.SimplexSearchOptimizer`
  simplex search (aka Nelder-Mead method)

These are Java class paths to `IOptimizer` implementations; this allows to extend range of methods easily. Please refer to Chapter **??** for a discussion and comparison of the methods. If no method is specified, the greedy algorithm is used by+- default.

Chapter 3

Integrating into User Code

By default, PATUS creates a C source file named `kernel.c` (The default setting can be overridden with the `--kernel-file` command line option, cf. Appendix 2.) This kernel file contains all the generated code variants of the stencil kernel, a function selecting one of the code variants, and an initialization function, `initialize`, which does the NUMA-aware data initialization and should preferably be called directly after allocating the data (cf. Chapter **??**).

Example 3.1: *The generated stencil kernel code for the example wave stencil.*

The generated stencil kernel and the data initialization function have the following signatures:

```
void wave (float** u_0_1_out,
  float* u_0_m1, float* u_0_0, float* u_0_1, float c2dt_h2,
  int N,
  int cb_x, int cb_y, int cb_z, int chunk, int _unroll_p3);

void initialize (
  float* u_0_m1, float* u_0_0, float* u_0_1, float dt_dx_sq,
  int N,
  int cb_x, int cb_y, int cb_z, int chunk);
```

The selector function, which is named exactly as the stencil in the stencil specification (`wave` in Ex. 3.1), is the function, which should be called in the user code. Its parameters are:

- pointers to the grid array containing the results; these are double pointers and marked with the `_out` suffix;

- input grid arrays, one for each time index required to carry out the stencil computation; e.g., in the wave equation example, three time indices are required: the result time step $t + 1$, which depends on the input time steps t and $t - 1$; the time index is appended to the grid identifier as last suffix; the `m` stands for "minus";
- any parameters defined in the stencil specification, e.g., `c2dt_h2` in the wave equation example;
- all the variables used to specify the size of the problem domain, only `N` in our example;
- Strategy- and optimization-related parameters, the best values of which were determined by the auto-tuner and can be substituted into the stencil kernel function call in the user code; the strategy used in the example has one cache block size parameter, (`cb_x`, `cb_y`, `cb_z`) and a chunk size `chunk`; furthermore, `_unroll_p3` determines the loop unrolling configuration.

The output pointers are required because PATUS rotates the time step grids internally, i.e., the input grids change roles after each spatial sweep. Thus, the user does typically not know which of the grid arrays contains the solution. A pointer to the solution grid is therefore assigned to the output pointer upon exit of the kernel function.

The initialization function can be modified to reflect the initial condition required by the problem. However, the generated loop structure should not be altered. Alternatively, a custom initialization can be done after calling the initialization function generated by PATUS.

Chapter 4

Alternate Entry Points to Patus

As the auto-tuner module is decoupled from the code generating system, it can be used as a stand-alone auto-tuner for other codes besides the ones created by PATUS. If the user has a hand-crafted existing parametrized code for which the best configurations need to be determined, the PATUS auto-tuner is a perfectly valid option to find the best set of parameters. The only requirements are that the tunable parameters must be exposed as command line parameters and that the program must print the timing information to `stdout` as last line of the program output in an arbitrary time measurement unit. Note that the auto-tuner tries to *minimize* this number, i.e., it must be a time measurement rather than a performance number.

Chapter 5

Current Limitations

In the current state, there are several limitations to the PATUS framework:

- Only shared memory architectures are supported (specifically: shared memory CPU systems and single-GPU setups).

- 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. Such a grid traversal is called a Jacobi iteration. In particular, this rules out schemes with special traversal rules such as red-black Gauss-Seidel iterations.

- No extra boundary handling is applied. The stencil is applied to every interior grid point, but not to boundary points. I.e., the boundary values are kept constant; this corresponds to Dirichlet boundary conditions. To implement boundary conditions, they could be factored into the stencil operation by means of extra coefficient grids. In the same way, non-rectilinear domain shapes and non-uniform grids can be emulated by providing the shape and/or geometry information encoded as coefficients in additional grids. Alternatively, special $(d-1)$-dimensional stencil kernels could be specified for the boundary treatment, which are invoked after the d-dimensional stencil kernel operating on the interior. This approach, however, will invalidate temporal blocking schemes.

- The index calculation assumes that the stencil computation is carried out on a flat grid (or a grid which is homotopic to a flat grid). In particular, currently no spherical or torical geometries are implemented, which require modulo index calculations.

- There is no support for temporally blocked schemes yet.

Bibliography

[1] M. Frigo and S. Johnson. The Design and Implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. Special issue on "Program Generation, Optimization, and Platform Adaptation". [cited at p. 6]

Appendices

Appendix A

Patus Grammars

A.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*⟩ ::= '+' | '-' | '*' | '/' | '^'

A.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*⟩ ::= '<' | '<=' | '==' | '>=' | '>' | '!='

Appendix B

Stencil Specifications

B.1 Basic Differential Operators

B.1.1 Laplacian

```
stencil laplacian
{
  domainsize = (1 .. N, 1 .. N, 1 .. N);
  t_max = 1;

  operation (float grid u, float param alpha, float param beta)
  {
    u[x, y, z; t+1] =
      alpha * u[x, y, z; t] +
      beta  * (
        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]);
  }
}
```

B.1.2 Divergence

```
stencil divergence
{
  domainsize = (1 .. x_max, 1 .. y_max, 1 .. z_max);
  t_max = 1;

  operation (
    float grid u(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    const float grid ux(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    const float grid uy(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
```

```
    const float grid uz(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    float param alpha, float param beta, float param gamma)
  {
    u[x, y, z; t] =
      alpha * (ux[x+1, y, z] - ux[x-1, y, z]) +
      beta  * (uy[x, y+1, z] - uy[x, y-1, z]) +
      gamma * (uz[x, y, z+1] - uz[x, y, z-1]);
  }
}
```

B.1.3 Gradient

```
stencil gradient
{
  domainsize = (1 .. x_max, 1 .. y_max, 1 .. z_max);
  t_max = 1;

  operation (
    const float grid u(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    float grid ux(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    float grid uy(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    float grid uz(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    float param alpha, float param beta, float param gamma)
  {
    ux[x, y, z; t] = alpha * (u[x+1, y, z] + u[x-1, y, z]);
    uy[x, y, z; t] = beta  * (u[x, y+1, z] + u[x, y-1, z]);
    uz[x, y, z; t] = gamma * (u[x, y, z+1] + u[x, y, z-1]);
  }
}
```

B.2 Wave Equation

```
stencil wave
{
    domainsize = (1 .. N, 1 .. N, 1 .. N);
    t_max = 100;

    operation (float grid u, float param c2dt_h2)
    {
        u[x, y, z; t+1] = 2 * u[x, y, z; t] - u[x, y, z; t-1] +
            c2dt_h2 * (
                -15/2 * u[x, y, z; t] +
                4/3 * (
                    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]
                )
                -1/12 * (
```

```
                    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]
                )
            );
    }
}
```

B.3 COSMO

B.3.1 Upstream

```
stencil upstream_5_3d
{
  domainsize = (1 .. x_max, 1 .. y_max, 1 .. z_max);
  t_max = 1;

  operation (double grid u, double param a)
  {
    u[x, y, z; t+1] = a * (
       -2 * (u[x-3, y, z; t] + u[x, y-3, z; t] + u[x, y, z-3; t]) +
       15 * (u[x-2, y, z; t] + u[x, y-2, z; t] + u[x, y, z-2; t]) +
      -60 * (u[x-1, y, z; t] + u[x, y-1, z; t] + u[x, y, z-1; t]) +
       20 *  u[x,   y, z; t] +
       30 * (u[x+1, y, z; t] + u[x, y+1, z; t] + u[x, y, z+1; t]) +
       -3 * (u[x+2, y, z; t] + u[x, y+2, z; t] + u[x, y, z+2; t]);
  }
}
```

B.3.2 Tricubic Interpolation

```
stencil tricubic_interpolation
{
  domainsize = (1 .. x_max, 1 .. y_max, 1 .. z_max);
  t_max = 1;

  operation (double grid u,
    const double grid a, const double grid b, const double grid c)
  {
    double w1_a = 1.0/6.0*a[x,y,z]*(a[x,y,z]+1.0)*(a[x,y,z]+2.0);
    double w2_a = -0.5*(a[x,y,z]-1.0)*(a[x,y,z]+1.0)*(a[x,y,z]+2.0);
    double w3_a =  0.5*(a[x,y,z]-1.0)*a[x,y,z]*(a[x,y,z]+2.0);
    double w4_a = -1.0/6.0*(a[x,y,z]-1.0)*a[x,y,z]*(a[x,y,z]+1.0);

    double w1_b = 1.0/6.0*b[x,y,z]*(b[x,y,z]+1.0)*(b[x,y,z]+2.0);
    double w2_b = -0.5*(b[x,y,z]-1.0)*(b[x,y,z]+1.0)*(b[x,y,z]+2.0);
    double w3_b =  0.5*(b[x,y,z]-1.0)*b[x,y,z]*(b[x,y,z]+2.0);
    double w4_b = -1.0/6.0*(b[x,y,z]-1.0)*b[x,y,z]*(b[x,y,z]+1.0);
```

```
    double w1_c = 1.0/6.0*c[x,y,z]*(c[x,y,z]+1.0)*(c[x,y,z]+2.0);
    double w2_c = -0.5*(c[x,y,z]-1.0)*(c[x,y,z]+1.0)*(c[x,y,z]+2.0);
    double w3_c =  0.5*(c[x,y,z]-1.0)*c[x,y,z]*(c[x,y,z]+2.0);
    double w4_c = -1.0/6.0*(c[x,y,z]-1.0)*c[x,y,z]*(c[x,y,z]+1.0);

    u[x, y, z; t+1] =
      w1_a * w1_b * w1_c * u[x-1, y-1, z-1; t] +
      w2_a * w1_b * w1_c * u[x,   y-1, z-1; t] +
      w3_a * w1_b * w1_c * u[x+1, y-1, z-1; t] +
      w4_a * w1_b * w1_c * u[x+2, y-1, z-1; t] +
      ... // etc. for all 64 combinations of w?_a * w?_b * w?_c
          // and u[x+d1, y+d2, z+d3; t],  d1,d2,d3=-1,0,1,2
  }
}
```

B.4 Hyperthermia

```
stencil hyperthermia
{
  domainsize = (1 .. x_max, 1 .. y_max, 1 .. z_max);
  t_max = 1;

  operation (
    float grid T(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1),
    const float grid c(0 .. x_max+1, 0 .. y_max+1, 0 .. z_max+1)[9])
  {
    T[x, y, z; t+1] =

      // center point
      T[x, y, z; t] * (c[x, y, z; 0] * T[x, y, z; t] + c[x, y, z; 1]) +
      c[x, y, z; 2] +

      // faces
      c[x, y, z; 3] * T[x-1, y, z; t] +
      c[x, y, z; 4] * T[x+1, y, z; t] +
      c[x, y, z; 5] * T[x, y-1, z; t] +
      c[x, y, z; 6] * T[x, y+1, z; t] +
      c[x, y, z; 7] * T[x, y, z-1; t] +
      c[x, y, z; 8] * T[x, y, z+1; t];
  }
}
```

B.5 Anelastic Wave Propagation

B.5.1 uxx1

```
stencil pmcl3d_uxx1
{
  domainsize = (nxb .. nxe, nyb .. nye, nzb .. nze);
  t_max = 1;

  operation (
    const float grid d1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid u1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid xx(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid xy(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid xz(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float param dth)
  {
    float c1 = 9./8.;
    float c2 = -1./24.;

    float d = 0.25 *
      (d1[x,y,z] + d1[x,y-1,z] + d1[x,y,z-1] + d1[x,y-1,z-1]);
    u1[x,y,z; t+1] = u1[x,y,z; t] + (dth / d) * (
      c1 * (
        xx[x,  y,z] - xx[x-1,y,z] +
        xy[x,y,  z] - xy[x,y-1,z] +
        xz[x,y,z  ] - xz[x,y,z-1]) +
      c2 * (
        xx[x+1,y,z] - xx[x-2,y,z] +
        xy[x,y+1,z] - xy[x,y-2,z] +
        xz[x,y,z+1] - xz[x,y,z-2])
    );
  }
}
```

B.5.2 xy1

```
stencil pmcl3d_xy1
{
  domainsize = (nxb .. nxe, nyb .. nye, nzb .. nze);
  t_max = 1;

  operation (
    const float grid mu(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid xy(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid u1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid v1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float param dth)
  {
    float c1 = 9. / 8.;
    float c2 = -1. / 24.;

    float xmu = 2. / (1. / mu[x, y, z] + 1. / mu[x, y, z-1]);
```

```
    xy[x, y, z; t+1] = xy[x, y, z; t] + dth * xmu * (
      c1 * (
        u1[x, y+1, z] - u1[x,   y, z] +
        v1[x, y,   z] - v1[x-1, y, z]
      ) +
      c2 * (
        u1[x,   y+2, z] - u1[x,   y-1, z] +
        v1[x+1, y,   z] - v1[x-2, y,   z]
      )
    );
  }
}
```

B.5.3 xyz1

```
stencil pmcl3d_xyz1
{
  domainsize = (nxb .. nxe, nyb .. nye, nzb .. nze);
  t_max = 1;

  operation (
    const float grid mu(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid lam(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid u1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid v1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid w1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid xx(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid yy(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid zz(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float param dth)
  {
    float c1 = 9./8.;
    float c2 = -1./24.;

    float b = 8. / (
      1. / lam[x, y,   z  ] + 1. / lam[x+1, y,   z  ] +
      1. / lam[x, y-1, z  ] + 1. / lam[x+1, y-1, z  ] +
      1. / lam[x, y,   z-1] + 1. / lam[x+1, y,   z-1] +
      1. / lam[x, y-1, z-1] + 1. / lam[x+1, y-1, z-1]
    );

    float a = b + 2. * 8. / (
      1. / mu[x, y,   z  ] + 1. / mu[x+1, y,   z  ] +
      1. / mu[x, y-1, z  ] + 1. / mu[x+1, y-1, z  ] +
      1. / mu[x, y,   z-1] + 1. / mu[x+1, y,   z-1] +
      1. / mu[x, y-1, z-1] + 1. / mu[x+1, y-1, z-1]
    );

    // find xx stress
```

```
xx[x, y, z; t+1] = xx[x, y, z; t] + dth * (
  a * (
    c1 * (u1[x+1, y, z] - u1[x,   y, z]) +
    c2 * (u1[x+2, y, z] - u1[x-1, y, z])
  ) +
  b * (
    c1 * (
      v1[x, y, z] - v1[x, y-1, z  ] +
      w1[x, y, z] - w1[x, y,   z-1]
    ) +
    c2 * (
      v1[x, y+1, z  ] - v1[x, y-2, z  ] +
      w1[x, y,   z+1] - w1[x, y,   z-2]
    )
  )
);

// find yy stress
yy[x, y, z; t+1] = yy[x, y, z; t] + dth * (
  a * (
    c1 * (v1[x, y,   z] - v1[x, y-1, z]) +
    c2 * (v1[x, y+1, z] - v1[x, y-2, z])
  ) +
  b * (
    c1 * (
      u1[x+1, y, z] - u1[x, y, z  ] +
      w1[x,   y, z] - w1[x, y, z-1]
    ) +
    c2 * (
      u1[x+2, y, z  ] - u1[x-1, y, z  ] +
      w1[x,   y, z+1] - w1[x,   y, z-2]
    )
  )
);

// find zz stress
zz[x, y, z; t+1] = zz[x, y, z; t] + dth * (
  a * (
    c1 * (w1[x, y, z  ] - w1[x, y, z-1]) +
    c2 * (w1[x, y, z+1] - w1[x, y, z-2])
  ) +
  b * (
    c1 * (
      u1[x+1, y, z] - u1[x, y,   z] +
      v1[x,   y, z] - v1[x, y-1, z]
    ) +
    c2 * (
      u1[x+2, y,   z] - u1[x-1, y,   z] +
      v1[x,   y+1, z] - v1[x,   y-2, z]
    )
  )
```

```
    );
  }
}
```

B.5.4 xyzq

```
stencil pmcl3d_xyzq
{
  domainsize = (nxb .. nxe, nyb .. nye, nzb .. nze);
  t_max = 1;

  operation (
    const float grid mu(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid lam(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid r1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid r2(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid r3(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid xx(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid yy(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float grid zz(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid u1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid v1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid w1(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid qp(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid qs(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    const float grid tau(-1 .. nxt+2, -1 .. nyt+2, -1 .. nzt+2),
    float param dh, float param dt, float param dth,
    float param nz)
  {
    float c1 = 9./8.;
    float c2 = -1./24.;

    float d = 8. / (
      1. / mu[x, y,   z  ] + 1. / mu[x+1, y,   z  ] +
      1. / mu[x, y-1, z  ] + 1. / mu[x+1, y-1, z  ] +
      1. / mu[x, y,   z-1] + 1. / mu[x+1, y,   z-1] +
      1. / mu[x, y-1, z-1] + 1. / mu[x+1, y-1, z-1]
    );

      float a2 = 2 * d;
    float c = a2 + 8. / (
      1. / lam[x, y,   z  ] + 1. / lam[x+1, y,   z  ] +
      1. / lam[x, y-1, z  ] + 1. / lam[x+1, y-1, z  ] +
      1. / lam[x, y,   z-1] + 1. / lam[x+1, y,   z-1] +
      1. / lam[x, y-1, z-1] + 1. / lam[x+1, y-1, z-1]
    );

    float qpa = 0.125 * (
      qp[x, y,   z  ] + qp[x+1, y,   z  ] +
      qp[x, y-1, z  ] + qp[x+1, y-1, z  ] +
```

```
      qp[x, y,   z-1] + qp[x+1, y,   z-1] +
      qp[x, y-1, z-1] + qp[x+1, y-1, z-1]
    );

    float qsa = 0.125 * (
      qs[x, y,   z  ] + qs[x+1, y,   z  ] +
      qs[x, y-1, z  ] + qs[x+1, y-1, z  ] +
      qs[x, y,   z-1] + qs[x+1, y,   z-1] +
      qs[x, y-1, z-1] + qs[x+1, y-1, z-1]
    );

    // (we can't handle indirect grid accesses for the time being)
    // float tauu = tau[((coords1 * nxt + this.x) % 2) +
    //     2 * ((coords2 * nyt + this.y) % 2) +
    //     4 * ((nz + 1 - (coords3 * nzt + this.z)) % 2)];

    float vxx = c1 * (u1[x+1, y, z] - u1[x, y, z]) +
      c2 * (u1[x+2, y, z] - u1[x-1, y, z]);
    float vyy = c1 * (v1[x, y, z] - v1[x, y-1, z]) +
      c2 * (v1[x, y+1, z] - v1[x, y-2, z]);
    float vzz = c1 * (w1[x, y, z] - w1[x, y, z-1]) +
      c2 * (w1[x, y, z+1] - w1[x, y, z-2]);

    float a1 = -qpa * c * (vxx + vyy + vzz) / (2. * dh);

    //float x1 = tauu / dt + 0.5;
    float x1 = tau[x, y, z] / dt + 0.5;
    float x2 = x1 - 1; // (tauu/dt)-(1./2.)

    // normal stress xx, yy and zz
    xx[x, y, z; t+1] = xx[x, y, z; t] + dth * (c * vxx + (c - a2) *
      (vyy + vzz)) + dt * r1[x, y, z; t];
    yy[x, y, z; t+1] = yy[x, y, z; t] + dth * (c * vyy + (c - a2) *
      (vxx + vzz)) + dt * r2[x, y, z; t];
    zz[x, y, z; t+1] = zz[x, y, z; t] + dth * (c * vzz + (c - a2) *
      (vxx + vyy)) + dt * r3[x, y, z; t];

    float hdh = -d * qsa / dh;
    r1[x, y, z; t+1] = (x2 * r1[x, y, z; t] - hdh*(vyy + vzz) + a1)/x1;
    r2[x, y, z; t+1] = (x2 * r2[x, y, z; t] - hdh*(vxx + vzz) + a1)/x1;
    r3[x, y, z; t+1] = (x2 * r3[x, y, z; t] - hdh*(vxx + vyy) + a1)/x1;

    xx[x, y, z; t+1] = xx[x, y, z; t+1] + dt * r1[x, y, z; t+1];
    yy[x, y, z; t+1] = yy[x, y, z; t+1] + dt * r2[x, y, z; t+1];
    zz[x, y, z; t+1] = zz[x, y, z; t+1] + dt * r3[x, y, z; t+1];
  }
}
```