Overview of the toolkit
Code generation allows you to write a PDE solver and automatically generate high performance implementations on CPU and GPU, including MPI. It is a necessary tool to achieve performance portability.

NAG has created a Domain Specific Language (DSL) embedded in C++. The DSL allows users to specify stencil based solvers on structured grids. A Clang tool parses the code and produces implementations targeting OpenMP+SIMD or CUDA.

**Toolkit functionality**
- Supports PDEs in up to 6 dimensions (and can go higher)
- Matrix-free iterative solvers (GMRES, CG, BiCGStab)
- Preconditioners, direct solvers and automatic matrix assembly
- Possible to implement finite element solvers on structured grids
- Operator splitting for ADI time steppers
- Highly optimised batch tridiagonal solvers
- Ability to use existing C/C++ code (with restrictions)
- Built-in algorithmic differentiation
- Batch solution of similar PDEs, e.g. risk calculations

A DSL embedded in C++
The toolkit is exposed as a standards compliant C++11 class library, with some additional “language rules.” Examples of additional rules:
- stencil accesses are only allowed inside Operators
- stencil offsets must be integer constant expressions (ICEs)
- only scalars, Dats or aggregates of these may be accessed inside Operators

The Clang tool enforces the DSL rules and handles code generation. Below are snippets from an example solving an SLV-type model with Hundsdorfer-Verwer time stepping. The full code is available on request.

```cpp
// Example of an existing user function
template<class FP>
void foo( ... /* user args */ ...) 
{ 
    using namespace pdetk;
    // ...
}
```

**Project status and early results**
The NAG PDE Toolkit is in early development with release slated for Q1’20. Results from current development builds solving the SLV problem outlined are shown below. The numbers include a certain amount of hand-tuning by NAG. The goal is to build NAG’s expertise into the PDE Toolkit code generation.

Grid size (v x v) is across the top, batch size is down the side. The solver performs 100 ADI time steps with all calculations in double precision. The table reports the number of PDEs per second (PVs/s).

<table>
<thead>
<tr>
<th>Grid Size</th>
<th>Intel Xeon W-2145 (Skylake), 8 cores</th>
<th>NVIDIA V100, 16GB RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2048</td>
<td>64 350 68 31 27 14 7</td>
<td>64 70 60 50 50 39 28</td>
</tr>
<tr>
<td>1536</td>
<td>96 380 68 31 27 14 7</td>
<td>96 110 85 66 65 46 32</td>
</tr>
<tr>
<td>1000</td>
<td>192 420 68 31 27 14 7</td>
<td>192 205 140 100 90 58 37</td>
</tr>
<tr>
<td>2048</td>
<td>320 450 68 31 27 14 7</td>
<td>320 330 180 120 110 66 40</td>
</tr>
<tr>
<td>1536</td>
<td>576 550 240 140 126 71 OOM</td>
<td>576 550 240 140 126 71 OOM</td>
</tr>
<tr>
<td>1000</td>
<td>992 850 280 153 137 OOM</td>
<td>992 850 280 153 137 OOM</td>
</tr>
<tr>
<td>2048</td>
<td>1536 1150 320 161 OOM</td>
<td>1536 1150 320 161 OOM</td>
</tr>
<tr>
<td>1000</td>
<td>2048 1380 340 OOM</td>
<td>2048 1380 340 OOM</td>
</tr>
</tbody>
</table>

“OOM” means the V100 ran out of memory. CPU code is vectorised and performance is stable. To fully load the GPU with small problems requires huge batch sizes. If problem size increases even modestly, GPU performance picks up strongly and continues to scale as the card is given more work.

**Early engagement**
If the functionality and performance outlined here interests you, then NAG would like to discuss your requirements in more detail to ensure the PDE Toolkit integrates smoothly into your environment.

For further information please contact support@nag.co.uk.