Improving Performance Portability and Exascale Software Productivity with the ∇ Numerical Programming Language

EASC2015 - Edinburg, Scotland
Exascale Applications and Software Conference

CAMIER Jean-Sylvain

CEA, DAM, DIF, F-91297 Arpajon, France

21~23 April 2015
Challenges and Objectives

Objectives & Roadmap since 2009:

- **Performances**: Instantiate the right programming model for different SW/HW stacks
- **Portability**: Provide portable scientific applications across architectures
- **Programmability**: Attractive approach for tomorrow’s SW engineers
- **Interoperability**: Allow modularity with legacy codes

Challenges for Application Developers:

- Abstractions:
  \[ i \, \Delta_n^1(\psi(n, i, j)) + \lambda \, E_n(\Delta_n^1(\psi(n, i, j))) = 0 \]
  \[ + E_j^{-1}(\Delta_j^2(\psi(n, i, j))) \]
  Schrödinger - 2D Finite Differences Scheme

- Tuning
- High Performances & Code Portability

Concurency, Vectorization, Data access, Locality, Cache hierarchies, Resiliency
## Numerical Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Application</th>
<th># of ∇ lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicative Unstructured (+scatter/gather)</td>
<td>LULESH (1.0) (LLNL)</td>
<td>1030</td>
</tr>
<tr>
<td>Explicative Structured</td>
<td>HYDRO (CEA)</td>
<td>757</td>
</tr>
<tr>
<td>Implicite</td>
<td>M-NL-DDFV (CEA/DAM) Schrödinger (CEA/DAM)</td>
<td>2304 / 375</td>
</tr>
<tr>
<td>Monte-Carlo</td>
<td>MCTB (CEA/DAM)</td>
<td>828</td>
</tr>
<tr>
<td>Dynamique Molecular</td>
<td>CoMD (LANL) MiniMD (SNL)</td>
<td>293 / 474</td>
</tr>
<tr>
<td>SPH</td>
<td>SPH (CEA/DAM)</td>
<td>2500</td>
</tr>
</tbody>
</table>

### Visual Representations

![LULESH](image1.png)  ![HYDRO](image2.png)  ![MNLDFFV](image3.png)  ![Schrödinger](image4.png)  ![CoMD](image5.png)  ![SPH](image6.png)
Strategy of the $\nabla$ Toolchain

Proxy-Apps

- App. **options**
- Data **variables**
- **For-Loops** focus
- **Constraints:** no glue-code '@' concurrency

$\nabla$ Howto:

Same $\nabla$ sources

Sources Analysis

Frontend Lex/Parse
Abstract Syntactic Tree (AST)
Analysis

Middleend Common
Optimizations

Optimizations & Transformations

Compilation command line

Multiple Back-Ends
(Code Generation)
Explicit declaration: **Libraries, Options** and mesh **Variables:**

```c
with cartesian;
// ≡ aleph ≡ Linear Algebra Extended
to Hybrid Parallelism

options{
    Real γ = 1.4;
    Real option_δt_fixed = 1.e-7;
    Integer option_max_iterations = 1024;
    Bool option_quads = true;
};
```

Data-parallelism is **implicitly** expressed via jobs **items:**

```c
//cells
void geomComputeQuadSurface(void)
out (cell cell_A) @ -10.0 if (option_quads){
    Real3 fst_edge = coord[2]-coord[0];
    Real3 snd_edge = coord[3]-coord[1];
    cell_A=\frac{1}{2}*(fst_edge@snd_edge);
}

//nodes
void geomComputeNodeArea(void)
in (cell cell_A) out (node node_A) @ -8.5{
    node_A=0.0;
    foreach cell node_A+=cell_A/nbNode;
}
```

```c
//faces
Real timeStepCtrl(Real local_δt)
in (cell absCQs,c,λ,V) out (cell glace_δt){
    Real Σljr=0.0;
    foreach node Σljr+=absCQs;
    glace_δt=2.0*λ*V/(c*Σljr);
    if(glace_δt<local_δt)
        local_δt = min(local_δt,glace_δt);
    return local_δt;
}
```

```
```
Jobs parallelism is **explicitly** declared via Hierarchical Logical Time (HLT)

- An additional dimension in concurrency is gained ⇒ **partial-ordering**!
- the ‘@’ statements ensure this new concurrency between all jobs

```c
cells void computeAQS(void)
    in (cell λ,ρ,c,CQS,absCQS)
    out (cell AQS) @ -1.0,16.0{
        Real pc = λ*ρ*c;
        foreach node{
            AQS = CQS@CQS;
            AQS *= pc/absCQS;
        }
    }

cells void computeStdFluxΣ(void)
    in (cell p,u,ũ,AQS,CQs)
    out (cell mfΣ, EfΣ) @ -∞,-1.0,π²{
        foreach node{
            Real3 Δu = u-ũ;
            Real3 FQs = AQS@Δu;
            FQs += p*CQs;
            mfΣ -= FQs;
            EfΣ -= FQs@ũ;
        }
    }
```

- Consistency and liveliness can be analysed and proved offline
- Instrumentation can be integrated in the framework
- Optimization and characterization stages can be again inserted before generation
  - loop fusion, data layout, prefetching, caches awareness, vectorization
own nodes void δNodes(void)
in (node node_is_on_∂Ω, node_Λ, node_θ, 
    face Cosθ, sdivs) @ 3.4{
Real δn,Σδ=0.0;
if (node_is_on_∂Ω) continue;
foreach face {
    Node other_node = (node[0]==this)?node[1]:node[0];
    δn=δ nodeList_Λ;
    Σδ+=1.0/(Cosθ*sdivs);
    // Extra-diagonal terms
    // DOFs ≡ (node_θ, this) and (node_θ, other_node)
    ∑ matrix addValue(node_θ, this, node_θ, other_node, -δn);
}
Σδ*=δt/node_Λ;
// Diagonal contribution, DOF ≡ (node_θ, this)^2
∑ matrix addValue(node_θ, this, node_θ, this, 1.0+Σδ);
}

void rhsInit(void) @ 1.0 {...}
own cells void setRhsCellValues(void) ... @ 1.1 {...}
own nodes void setRhsNodeValues(void) ... @ 1.1 {...}
own outer faces void setRhsFaceValues(void) ... @ 1.1 {...}
own outer faces void setStdRhsEdgesOnBorders(void) ... @ 1.2 {...}
own cells void alphaCells(void) ... @ 3.0 {...}
own cells void betaCells(void) ... @ 3.0 {...}
own nodes void gammaNodes(void) ... @ 3.0 {...}
own nodes void deltaNodes(void) ... @ 3.0 {...}
own outer faces void dirichletFaces(void) ... @ 3.0 {...}
own nodes void dirichletNodes(void) ... @ 3.0 {...}
void assembleAndSolve(void) @ 4.0 {...}
own cells void getCellResults(void) ... @ 4.3 {...}
own nodes void getNodeResults(void) ... @ 4.3 {...}
own cells void saveCellDensity(void) ... @ 4.4 {...}
own nodes void saveNodeDensity(void) ... @ 4.4 {...}
Hierarchical Logical Time: Presentation on $\nabla$ LULESH

Table: \(\nabla\) LULESH Logical Time (\(\neq\) Point Of Views)

1 - Logical Timeline of $\nabla$ LULESH jobs

2 - Timeline $\Rightarrow$ Swirl-loop P.o.V

3 - Logically-timed Component P.o.V
Hierarchical Logical Time: Nested Composition

- **Application** ≡ **Nested Composition** of such logically-timed **components**

- Each **component** or **entity** is instanciated hierarchically
  - Now via command-line
  - ⇒ The need of front-end tools will rapidly be crucial as applications will grow bigger!
Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics[1]

- 3k sloc serial programming model vs 1k sloc w/o comments
  - Same average ratio for other programs

Different backends are directly available:

- **ARCANE** [2]: with Threads, MPI, MPI+Threads, MPC [3]
- **CUDA** [NVIDIA]: still naive, full-GPU, single-node code generation
- **OKINA**: Stand-alone C/C++ native programming model
  - (OpenMP or Cilk+) with no-vec, SSE, AVX, AVX512 or MIC
  - First try with full intrinsics and gather/scatter generated instructions

Yours...
\textbf{V-LULESH} Comparative Test on \textbf{Intel® Xeon Sandy Bridge™}

\textbf{V-Lulesh Comparative Performance Test with OpenMP on Intel® Xeon Sandy Bridge E5-2680-2.70GHz}

Cell updates per us

- Cores for 32$^3$ Cells
- Cores for 48$^3$ Cells
- Cores for 64$^3$ Cells
- Cores for 96$^3$ Cells
$\nabla$-LULESH Comparative Test on Intel® Xeon Haswell™

$\nabla$-Lulesh Best vs OKINA+OpenMP Speedup
on one Intel® Xeon™ CPU E3-1240 v3 3.40GHz (Quad Core)
\textbf{\textit{V-LULESH} Comparative Test on Intel® XeonPhi™}

\begin{center}
\textbf{V-Lulesh Comparative Performance Test with OpenMP on Intel® Phi™ at 1.052GHz}
\end{center}

- Cell updates per ms
- Threads for $32^3$ Cells
- Threads for $48^3$ Cells
- Threads for $64^3$ Cells
- Threads for $96^3$ Cells

- MIC-ref.
- MIC-V
- MIC-V
∇-LULESH Speedup on single Intel® XeonPhi™

∇-Lulesh Speedup on one Intel® Xeon Phi™

nb Cells

nb Threads

Speedup

CEA/DAM/DIF/DSSI | 21~23 April 2015 | PAGE 14/18
Conclusion & Roadmap

Conclusion

∇ DSL work-in-progress ⇒ Raise Loop-Level’s abstraction!
- No need to choose today the best programming model for tomorrow’s architectures
- Does not require to code multiple versions of kernels for different models

∇ Helps transition from Bulk-Synchronous-Programming (BSP)
- Performances: Allowing specific optimization phases before code generation
- Portability: Adaptable towards diversified & complex emerging architectures
- Productivity: Simple, Attractive & Elegant

∇ CeCILL v2.1 license, under French law ⇒ www.nabla-lang.org
- The CeCILL is a free software license, explicitly compatible with the GNU GPL

Backends & Roadmap (~2017)

∇ Lawrence Livermore National Laboratory: RAJA Portability Layer, Los Alamos National Laboratory: Kokkos, Legion
∇ CEA: Okina (MPC, OpenMP, Cilk+)
∇ NVIDIA: CUDA, Chapel

Proxy applications: ports and performance evaluations
Presentation

The \( \nabla \) domain-specific language (DSL) provides a productive development way for exascale HPC technologies, flexible enough to be competitive in terms of performances.

This software is a computer program whose purpose is to translate specific numerical-analysis sources and generate optimized code for different targets and architectures.

Latest Tarball

<2015-02-18 Wed> Prototype #150218 released

This is the first release, with \( \nabla \) Lulesh as the main example. The \( \nabla \) compiler can generate sources from \( \nabla \) LULESH to three different backends:

- **Arcane**: our CEA's middleware,
- **Cuda**,
- **Okina**: a C/C++ stand-alone fully vecorized backend for standard compilers with OpenMP, Cilk+ or MPC.

Several other tests are given as examples:

