TALC: A Simple C Language Extension For Improved Performance and Code Maintainability

Jeff Keasler
Linux Cluster Institute
NCSA 2008
LLNL-PRES-403247
Prepared by LLNL under Contract DE-AC52-07NA27344
Lawrence Livermore National Laboratory
The Software Chasm

Many important HPC applications cannot be re-written for practical reasons:

- Size of code.
- Additional efforts of validating a rewrite.
- What do you rewrite to?
- Impact on budget and deliverables.

Moore’s law is now spurring a renaissance of architectural diversity in the HPC marketplace:

- Multicore (Intel, AMD, Sun, …)
- System on a chip (IBM BlueGene, SciCortex, …)
- The re-emergence of vector (Cray, ClearSpeed, Intel, …)
- Graphical Units to supplement work (IBM/Sony Cell, Nvidia, …)

What works well for one may work poorly for another.
Many software options exist to port to new architectures

- A *manual* rewrite would lock in one solution
- *Automated* transformations can generate each solution
We Would Like To Leverage a Single Source Code for Many Architectures

- Optimization Guidance (optional)
- Source Code (C/C++/Fortran)
- Coding Standards (optional)
- Transformations rules
- ROSE Compiler (CASC/Quinlan)
- Memory Management
- Architecture 1
- Architecture N
- Vendor Compiler
- Executable

Research
Existing
Generated
Motivation

- Many large science applications achieve a small fraction of peak performance

- Known roadblocks to performance include
  - User choice of data structures
  - Conservative optimization choices by compilers

- We are working on a source-to-source translator called TALC to allow users to control these issues for mesh based codes without modifying their source code
Mesh Based Physics

- Meshes are used to solve partial differential equations.
- Meshes are often described as a hierarchy of locality contexts.
- Examples of contexts include: subdomains, patches, finite elements and material regions.
- Data layout choices are often made within locality contexts to increase cache performance.

2D Mesh

Mesh

Elements

Nodes

Concrete

Copper
Fundamental Data Layouts

- **Array-Like**
  - `double x[10000] ;
    double y[10000] ;
    double z[10000] ;`

- **Struct-Like**
  - `struct coord {
      double x, y, z ;
    } point[10000] ;`

- **Clustered-Struct**
  - `struct coord {
      double x, y ;
    } point[10000] ;
    double z[10000] ;`

Memory Interleave

```
+---+---+---+---+---+---+---+---+---+---+---+
| x | x | x | x | x | x |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
| y | y | y | y | y | y |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
| z | z | z | z | z | z |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
+---+---+---+---+---+---+---+---+---+---+---+
| x | y | z | x | y | z |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
| x | y | x | y | x | y |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
| z | z | z | z | z | z |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+
```
double quarterDelta = 0.25 * deltaTime;

for (int i = 0 ; i < material_length ; i++){
    int index = material_map[i];
    double szz = - sxx[index] - syy[index] ;

    deltz[index] += quarterDelta * (vnew[index] + v[index]) *
        ( dxx[index] * (sxx[index] + newSxx[i]) +
          dyy[index] * (syy[index] + newSyy[i]) +
          dzz[index] * (szz + newSzz[i]) +
          2.*dxy[index] * (txy[index] + newTxy[i]) +
          2.*dyz[index] * (tyz[index] + newTyz[i]) ) ;

    delts[i] += quarterDelta * (vnew[index] + v[index]) *
        ( dxx[index] * sxx[index] +
          dyy[index] * syy[index] +
          dzz[index] * szz +
          2.*dxy[index] * txy[index] +
          2.*dxz[index] * txz[index] +
          2.*dyz[index] * tyz[index] ) ;
}

Here, each field variable occupies a separate array
for (int i = 0 ; i < material_length ; i++){
    int index = material_map[i];
    double szz = - elem[index].sxx – elem[index].syy ;

    elem[index].deltz += quarterDelta * (elem[index].vnew + elem[index].v) * 
    (   elem[index].dxx * (elem[index].sxx + materialElem[i].newSxx) + 
        elem[index].dyy * (elem[index].syy + materialElem[i].newSyy) + 
        elem[index].dzz * (       szz + materialElem[i].newSzz) + 
    2.*elem[index].dxy * (elem[index].txy + materialElem[i].newTxy) + 
    2.*elem[index].dxz * (elem[index].txz + materialElem[i].newTxz) + 
    2.*elem[index].dyz * (elem[index].tyz + materialElem[i].newTyz ) ) ;

    materialElem[i].dels += quarterDelta * (elem[index].vnew + elem[index].v) * 
    (   elem[index].dxx * elem[index].sxx + elem[index].dyy * elem[index].syy + 
        elem[index].dzz * szz + 2.*elem[index].dxy * elem[index].txy + 
    2.*elem[index].dxz * elem[index].txz + 2.*elem[index].dyz * elem[index].tyz ) ;
}

**Here, there are two contexts – mesh elements and material elements**
Stress-Strain Work Example – Clustered-Struct Layout

for (int i = 0 ; i < material_length ; i++){
    int index = material_map[i];
    double szz = - stress[index].sxx - stress[index].syy;

    deltz[index] += quarterDelta * (volume[index].vnew + volume[index].v) * 
        (deform[index].dxx * (stress[index].sxx + materialStress[i].newSxx) + 
        deform[index].dyy * (stress[index].syy + materialStress[i].newSyy) + 
        deform[index].dzz * (szz + materialStress[i].newSzz) + 
        2.*deform[index].dxy * (stress[index].txy + materialStress[i].newTxy) + 
        2.*deform[index].dxz * (stress[index].txz + materialStress[i].newTxz) + 
        2.*deform[index].dyz * (stress[index].tyz + materialStress[i].newTyz) ) ;

    delts[i] += quarterDelta * (volume[index].vnew + volume[index].v) * 
        (deform[index].dxx * stress[index].sxx + deform[index].dyy * stress[index].syy + 
        deform[index].dzz * szz + 2.*deform[index].dxy * stress[index].txy + 
        2.*deform[index].dxz * stress[index].txz + 2.*deform[index].dyz * stress[index].tyz ) ;
}

Here, contexts are created for each tightly bound group of field arrays
A mesh of 12000 elements contains two sparse material subsets of 8000 and 4000 elements. The 8000 element subset is evaluated...
### Stress-Strain Work Example – Cache Performance

<table>
<thead>
<tr>
<th>Data Layout</th>
<th>Hit Count</th>
<th>Miss Count</th>
<th>Hit Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array-Like</td>
<td>3955732080</td>
<td>286239697</td>
<td>93.3%</td>
</tr>
<tr>
<td>Intermediate</td>
<td>2842569424</td>
<td>281404535</td>
<td>91.0%</td>
</tr>
<tr>
<td>Struct-Like</td>
<td>2769568352</td>
<td>273753504</td>
<td>91.1%</td>
</tr>
</tbody>
</table>

Some applications/architectures optimize best with struct-like data layouts due to reduced *register pressure* or better use of *prefetch streams*.
Second Example – Quadrilateral Volume

- An unstructured mesh is created for quadrilaterals
  - Lattice of nodes stored as X and Y coordinate arrays
  - Quadrilateral shape defined by four arrays of nodal indices
- Wall clock run time is measured while varying
  - Compilers
  - Data representations (restricted pointers vs. STL)
  - Data Layouts
    - Separate coordinate and shape contexts
    - Switch between Array-Like and Struct-Like layout for each context
Quadrilateral Volume Example – Performance

GNU sees good optimizations for the first and third layout, while PathScale sees good optimizations for the second and fourth.
Quadrilateral Volume Example – Performance

GNU sees good optimizations for the first and third layout, while PathScale sees good optimizations for the second and fourth.

PGI sees more optimizations when using the STL. Note that Pathscale runs 25% slower when using the STL for data layouts number two and four.
Struct-Like Layouts Are Not Optimal For All Architectures

An x86 SSE enabled processor can optimize well with unaliased aligned array-like data

- `double *x = new double[10000] ;`
- `double *y = new double[10000] ;`
- `double *z = new double[10000] ;`

- Additional compiler directives are needed throughout the source code to indicate pointers are aligned
Memory Alignment Is Important For Many Architectures

**BG/L memory throughput:** \( a[i] = b[i] + ss \cdot c[i] \)

<table>
<thead>
<tr>
<th>Array Size</th>
<th>Unaligned(MB/s)</th>
<th>Aligned(MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>3040</td>
<td>6300</td>
</tr>
<tr>
<td>1000</td>
<td>3340</td>
<td>8270</td>
</tr>
<tr>
<td>10000</td>
<td>1290</td>
<td>3720</td>
</tr>
<tr>
<td>100000</td>
<td>1290</td>
<td>3720</td>
</tr>
<tr>
<td>500000</td>
<td>1290</td>
<td>1830</td>
</tr>
<tr>
<td>1000000</td>
<td>1280</td>
<td>1440</td>
</tr>
</tbody>
</table>

Results: Norris, Hartono, Gropp
Compiler Directives

- Memory Alignment
  - Library calls such as posix_memalign()
  - Compile line options such as `-Mcache_align`
  - Compiler directives such as
    - `__alignx()`
    - `declspec(alignment())`
    - `__attribute(alignment())`
    - `__assume_aligned()`

- Alias control
  - For C/C++ use restrict or `__restrict__`
Roadblocks to Data Layout Flexibility

- Users usually must rewrite their software to switch between Array-Like and Struct-Like data layouts or to take advantage of compiler directives
  - This makes it difficult to adapt software to compensate for performance idiosyncrasies of different compilers or memory subsystems
  - Software ends up being tuned for a specific hardware platform and compiler environment

- Dynamic Memory Management is often supported as a library rather than an integral part of the compiler
  - Compiler cannot generate aggressive optimizations due to incomplete knowledge of data layout, memory alignment, and inter-relations among heap pointers
TALC

- TALC is a source-to-source translator that allows users to direct compiler optimizations through the use of a schema file.

- The schema file provides a higher level of type information about the problem being solved.

- This enables a tight coordination between run-time memory allocation and compile-time code generation, which are currently somewhat disjoint.
TALC – Allowing User Directed Compiler Optimizations

- The Schema file contains high level information about data layouts

**Quadrilateral Schema 1**

```
View nodes
  Field x
  Field y
View
View elems
  Relation:nodes n1 n2 n3 n4
View
```

**Quadrilateral Schema 2**

```
View nodes
  Field x y
View
View elems
  Relation:nodes n1 n2 n3 n4
View
```
TALC Schema

Shock Tube Schema

- View mesh
  - View elems
    - Field mass momentum energy
    - Field pressure
  - View tube
    - Relation: faces upWindFace downWindFace
  - View
  - View
  - View faces
    - Field flux0 flux1 flux2
    - Relation: elems upWindElem downWindElem
  - View
  - View
In addition to controlling data layouts via a schema file, source-to-source translation allows us to
- Align variables when they are allocated on heap
- Apply machine specific compiler directives to indicate cache alignment and alias restriction

Features that allow this to work for us
- Consistent naming of Field arrays and contexts
- Hierarchical nature of context allocation already in place in many of our scientific codes
- Intimate familiarity with the structure of our codes
Potential Roadblocks

- Libraries
  - Most libraries expect passed arrays to have a specific memory layout (i.e. stride one array)
  - Even if compiling library source code, the user would need to understand the structure of the library software to create an appropriate schema

- I/O
  - Since many I/O operations are implemented using libraries, the same problem applies as above.
  - A library like MPI that provides a memory layout interface may be automatically transformable
Future Work

- Demonstration of Rapidmind backend
  - Will work on select loops at first, low performance

- Full thread support
  - Demonstration capability is already there

- Structured Indexsets

```c
MVmul(is *vecSpace, PntrR y, A, x) {
    while(vecSpace->("row")) {
        y = 0.0 ;
        while(vecSpace->("col")) {
            y += A*x ;
        }
    }
}
```
Conclusion

- A diversity of hardware architectures are being introduced simultaneously (Multi-core, NUMA, GPGPU/vector coprocessors)

- A low-impact change in our programming model may provide a unified way of running effectively on a diversity of system architectures

- A data-layout compiler has been written to explore this issue using the ROSE source-to-source translator