Loop Chaining: A Programming Abstraction For Balancing Locality and Parallelism

HIPS 2013 – Boston, MA

Christopher D. Krieger  Michelle Mills Strout  Andrew Stone
Catherine Olschanowsky  Stephen Guzik  Xinfeng Gao
Paul H.J. Kelly  Carlo Bertolli  Gihan Mudalige
Brian Van Straalen  Sam Williams
Modularity vs Performance

- Example of modularity in scientific codes
  - Iterate over all elements in set
  - Apply operator to each element using neighboring values
  - Keep iteration and each operator separate

- Performance is typically bound by memory bandwidth

- Tradeoff between modularity and locality
  - Keeping operators separate hurts locality by spacing out data reuse
int edgesToCells[numEdges][2]; // maps from edge to cells on either side

// for each cell, compute the change in area based on flow q
for (int cell=0; cell < numCells; ++cell)
    updateArea(q[cell], area_dt[cell]);

// for each edge, update the flux residual for cells on either side of edge
for (int edge=0; edge < numEdges; ++edge)
    calcFluxResidual(q[edges2Cells[0]], q[edges2Cells[1]],
                     area_dt[edges2Cells[0]], area_dt[edges2Cells[1]],
                     residual[edges2Cells[0]], residual[edges2Cells[1]]);

// update the flow field per cell based on the area and residual
for (int cell=0; cell < numCells; ++cell)
    updateFlowField(area_dt[cell], residual[cell], q[cell]);
Simple Loop Fusion Example

double total=0.0;
double A[N];

for (int i=0; i < N; ++i) {
    total += sin(A[i]);
}
for (int i=0; i < N; ++i) {
    total += cos(A[i]);
}
double total=0.0;
double A[N];

for (int i=0; i < N; ++i) {
    total += sin(A[i]);
}
for (int i=0; i < N; ++i) {
    total += cos(A[i]);
}
Simple Loop Fusion Example?

```c
double total=0.0;
double A[N];

for (int i=0; i < N; ++i) {
    total += sin(A[i]);
}
for (int i=0; i < N; ++i) {
    total += cos(A[i]);
}
```

gcc 4.8.1 –O3?  No
clang 3.3 –O3?  No
double total=0.0;
double A[N];

for (int i=0; i < N; ++i) {
    total += sin(A[i]);
}
for (int i=0; i < N; ++i) {
    total += cos(A[i]);
}
double total=0.0;
double A[N];

for (int i=0; i < N; ++i) {
    total += sin(A[i]);
}
for (int i=0; i < N; ++i) {
    total += cos(A[i]);
}

None of these production compilers performs loop fusion on this example.
Simple Loop Fusion Example?

double total=0.0;
double A[N];

for (int i=0; i < N; ++i) {
    total += sin(A[i]);
}
for (int i=0; i < N; ++i) {
    total += cos(A[i]);
}

None of these production compilers performs loop fusion on this example.

What do they lack?
Lack of information stymies optimization

- Compilers cannot apply some loop transformations due to:
  - Imperfect pointer and alias analysis
  - Limited side effect analysis

- We propose an abstraction that encapsulates the needed information.

- This abstraction is a loop chain.
A loop chain is a sequence of loops that share data

Restrictions
- Loops must be doall or reductions
- No intervening code between loops
Loop Chain Programming Abstraction

- Well defined iteration spaces for each loop, \( L_1, L_2, \ldots, L_N \)
- Data spaces for data accessed \( D_0, D_1, \ldots, D_M \)
- Access relations for data read/written by each iteration of each loop:
  - \( R_{ij} : L_i \rightarrow D_j \) or \( W_{ij} : L_i \rightarrow D_j \)
- Code for each loop body
Loop Chain View

Loop 1

Loop 2

Uodd

Loop 1

Ueven

Loop 2
Loop Chains Yield POSets

Loop Chain View

Partial Ordering
Related Solutions to Data Locality vs. Parallelism Tradeoff

- **Automatic systems**
  - Sequence of partitionable loops at OSU in Saday’s group
    - Pro: Data locality improvements between loops
    - Con: Compiler has to determine certain expressions are side-effect free
  - OpenMP to MPI conversion at Purdue in Eigenmann group
    - Pro: Use OpenMP pragmas to detect parallelism
    - Con: Compiler/run-time system must determine data access pattern

- **MPI**
  - Pro and Con: Locality is explicitly determined by programmer

- **PGAS languages, Chapel, X10**
  - Pro: Provide abstractions for associating data&comp to virtual processors
  - Con: Programmer has to determine how to aggregate data&comp

- ** Concurrent Collections (CnC) programming model**
  - Pro: Slicing annotations are an example of explicit access relations
  - Con: Programmer has to determine how to aggregate data&comp

- **Task Graph programming models**
  - Pro and Con: Require programmer to break work into tasks
Outline

- The Motivation Behind Loop Chains
- Definition of Loop Chains
- A Complete Example: Jacobi Solver
- Ways of Expressing Loop Chains
- Conclusion
Example: Jacobi Solver

Solve system of equations:

\[ Au = f \]

using a stationary iterative method

\[
u_i^{(k)} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} \ast u_j^{(k-1)} \right)
\]
Jacobi Code, N iterations

for (k=0; k < N; k++) {

    for (int i=1; i<=numrows; i++) {

        \[ u_i^{(k+1)} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} \times u_j^{(k)} \right) \]

    }

}
for (k=0; k < N ; k+=2) {

    for (int i=1; i<=numrows; i++) {
        $u_i^{even} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} \times u_j^{odd} \right)$
    }

    for (int i=1; i<numrows; i++) {
        $u_i^{odd} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} \times u_j^{even} \right)$
    }

}
Loops or Iteration Spaces

for (k=0; k < N ; k+=2) {

for (int i=1; i<=numrows; i++) {

\[ u_i^{even} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} * u_j^{odd} \right) \]

}

for (int i=1; i<=numrows; i++) {

\[ u_i^{odd} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} * u_j^{even} \right) \]

}

}
for (k=0; k < N ; k+=2) {
  for (int i=1; i<=numrows; i++) {
    $u_{i}^{even} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} \times u_{j}^{odd} \right)$
  }
  for (int i=1; i<=numrows; i++) {
    $u_{i}^{odd} = \frac{1}{A_{ii}} \left( f_i - \sum_{j \neq i} A_{ij} \times u_{j}^{even} \right)$
  }
}
Data Access Relations

for (k=0; k < N ; k+=2) {

for (int i=1; i<=numrows; i++) {

\[ u_{i}^{\text{even}} = \frac{1}{A_{ii}} \left( f_{i} - \sum_{j \neq i} A_{ij} * u_{j}^{\text{odd}} \right) \]

}

for (int i=1; i<=numrows; i++) {

\[ u_{i}^{\text{odd}} = \frac{1}{A_{ii}} \left( f_{i} - \sum_{j \neq i} A_{ij} * u_{j}^{\text{even}} \right) \]

}

\[
\begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\end{bmatrix}
\]

Loop 1

Loop 2

Uodd

Ueven
Partial Ordering of Jacobi

Loop Chain View

Partial Ordering
Scheduling Opportunities
Performance Improvements

Speedup of Jacobi Solvers Relative to Serial Implementation
thermal2 matrix, 4000 iterations, 880 tiles

Threads

0 5 10 15 20 25 30 35 40

Speedup over Serial

Blocked
Full Sparse Tiled
Outline

- The Motivation Behind Loop Chains
- Definition of Loop Chains
- A Complete Example: Jacobi Solver
- Ways of Expressing Loop Chains
- Conclusion
How Does Programmer Express a Loop Chain?

```
#pragma chainedloop chain1
iterspace(range(0, numrows-1)),
access(read, U, range(IA[i]<=p<IA[i+1]), JA[p])
access(readwrite, Uprime, identity, i)
```
OP2: Unstructured Mesh DSL

- OP2 is a DSL for specifying computation on unstructured meshes
- Allows programmer to specify a mesh and then iterate over parts of the mesh
  - For all edges, for all vertices, for all cells, etc.
- Programmer also explicitly specifies data spaces and maps between iteration and data spaces
call op_decl_set (edges, meshFile, "edges")
call op_decl_set (vertices, meshFile, "vertices")

call op_decl_dat (vertices, 6, vertexData, meshFile, "vertexData")
call op_decl_dat (edges, 6, edgeData, meshFile, "edgeData")

call op_decl_map (edges, vertices, 2, edges2Vertices, meshFile, "edges2Vertices")

call op_par_loop (edges, incrementKernel, &
op_arg_dat (vertexData, 1, edges2vertices, OP_INC), &
op_arg_dat (vertexData, 2, edges2vertices, OP_INC), &
op_arg_dat (edgeData, -1, OP_ID, OP_READ))
Expressing Loop Chains: OP2 – Unstructured Mesh DSL

call op_decl_set (edges, ...)
call op_decl_set (vertices, ...)
call op_decl_dat (vertices, 6, vertexData, meshFile, "vertexData")
call op_decl_dat (edges, 6, edgeData, meshFile, "edgeData")
call op_decl_map (edges, vertices, 2, edges2Vertices, meshFile, "edges2Vertices")
call op_par_loop (edges, incrementKernel, &
op_arg_dat (vertexData, 1, edges2vertices, OP_INC), &
op_arg_dat (vertexData, 2, edges2vertices, OP_INC), &
op_arg_dat (edgeData, -1, OP_ID, OP_READ))
Expressing Loop Chains:
OP2 – Unstructured Mesh DSL

call op_decl_set (edges, meshFile, "edges")
call op_decl_set (vertices, meshFile, "vertices")

call op_decl_dat (vertices, 6, vertexData, …)
call op_decl_dat (edges, 6, edgeData, …)

call op_decl_map (edges, vertices, 2, edges2Vertices, meshFile, "edges2Vertices")

call op_par_loop (edges, incrementKernel, &
op_arg_dat (vertexData, 1, edges2vertices, OP_INC), &
op_arg_dat (vertexData, 2, edges2vertices, OP_INC), &
op_arg_dat (edgeData, -1, OP_ID, OP_READ))
Expressing Loop Chains: 
OP2 – Unstructured Mesh DSL

call op_decl_set (edges, meshFile, "edges")
call op_decl_set (vertices, meshFile, "vertices")

call op_decl_dat (vertices, 6, vertexData, meshFile, "vertexData")
call op_decl_dat (edges, 6, edgeData, meshFile, "edgeData")

call op_decl_map (edges, vertices, 2, edges2Vertices, …)

call op_par_loop (edges, incrementKernel, &
op_arg_dat (vertexData, 1, edges2vertices, OP_INC), &
op_arg_dat (vertexData, 2, edges2vertices, OP_INC), &
op_arg_dat (edgeData, -1, OP_ID, OP_READ))
Expressing Loop Chains: OP2 – Unstructured Mesh DSL

```plaintext
call op_decl_set (edges, meshFile, "edges")
call op_decl_set (vertices, meshFile, "vertices")

call op_decl_dat (vertices, 6, vertexData, meshFile, "vertexData")
call op_decl_dat (edges, 6, edgeData, meshFile, "edgeData")

call op_decl_map (edges, vertices, 2, edges2vertices, meshFile, "edges2Vertices")

call op_par_loop (edges, incrementKernel, &
op_arg_dat (vertexData, 1, edges2vertices, OP_INC), &
op_arg_dat (vertexData, 2, edges2vertices, OP_INC), &
op_arg_dat (edgeData, -1, OP_ID, OP_READ))
```
begin_loopchain ()

! loop over edges
call op_par_loop (edges, incrementKernel, &
  op_arg_dat (vertexData, 1, edges2vertices, OP_INC), &
  op_arg_dat (vertexData, 2, edges2vertices, OP_INC), &
  op_arg_dat (edgeData, -1, OP_ID, OP_READ))

! loop over cells ...

! loop over vertices
call op_par_loop (vertices, kernel3, &
  op_arg_dat (temp, -1, OP_ID, OP_READ))
  op_arg_dat (q, -1, OP_ID, OP_WRITE))

end_loopchain ()
Outline

- The Motivation Behind Loop Chains
- Definition of Loop Chains
- A Complete Example: Jacobi Solver
- Ways of Expressing Loop Chains
- Conclusion
Loop chains are a programming abstraction that:

- Expose opportunities for cross-loop optimization to the compiler/runtime system
- Can be added to existing code in an incremental fashion while preserving modularity
- Require little effort on the part of the programmer

Loop chains are a natural extension to existing DSLs

Chombo, OP2 loop chaining work is underway
Thank You