A New, Portable Algorithm Framework for Parallel Linear Recurrence Problems

Wen-mei W. Hwu

with

Tom Jablin, Liwen Chang, Chris Rodrigues, Steven ShengZhou Wu, Abdul Dakkak

University of Illinois at Urbana-Champaign
Two Current Challenges for Petascale GPU Computing

- At scale use of GPUs
  - Communication costs dominate beyond 2048 nodes
  - E.g., NAMD Limited by PME
  - Insufficient computation work
- Programming Efforts
  - This talk
Writing efficient parallel code is complicated.
Tools can provide focused help or broad help

Planning how to execute an algorithm
Implementing the plan

• Choose data structures

• Map work/data to tasks
• Schedule tasks to threads

• Memory allocation
• Data movement

• Pointer operations
• Index arithmetic

GMAC
DL

Triolet, X10, Chappel, Nesl, DeLite, Par4All

• Kernel dimensions
• Thread ID arithmetic
• Synchronization
• Temporary data structures

OpenACC/C++AMP/Thrust

Tangram

Joint Lab 2013
Levels of GPU Programming Languages

<table>
<thead>
<tr>
<th>Prototype &amp; in development</th>
<th>X10, Chapel, Nesl, Delite, Par4all, Triolet...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implementation manages GPU threading and synchronization invisibly to user</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Next generation</th>
<th>OpenACC, C++AMP, Thrust, Bolt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplifies data movement, kernel details and kernel launch</td>
<td></td>
</tr>
<tr>
<td>Same GPU execution model (but less boilerplate)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current generation</th>
<th>CUDA, OpenCL, DirectCompute</th>
</tr>
</thead>
</table>
Where should the smarts be for Parallelization and Optimization?

• General-purpose language + parallelizing compiler
  – Requires a very intelligent compiler
  – Limited success outside of regular, static array algorithms

• Domain-specific language + domain-specific compiler
  – Simplify compiler’s job with language restrictions and extensions
  – Requires customizing a compiler for each domain

• Parallel meta-library + general-purpose compiler
  – Library embodies parallelization policies and decisions
  – Uses a general-purpose compiler infrastructure
  – Extensible—just add library functions
  – Historically, library is the area with the most success in parallel computing
Triolet – Composable Library-Driven Parallelization

• Allows library to collect multiple parallel operations and create an optimized arrangement
  – **Lazy evaluation** and aggressive inlining
  – **Loop fusion** to reduce communication and memory traffic
  – **Array partitioning** to reduce communication overhead
  – Library source-guided **parallelism optimization** of sequential, shared-memory, and/or distributed algorithms

• Loop-building decisions use information that is often known at compile time
  – By adding static typing to Python
def correlation(xs, ys):
    scores = (f(x,y) for x in xs for y in ys)
    return histogram(100, par(scores))

Compute $f(x,y)$ for every $x$ in $xs$ and for every $y$ in $ys$ (Doubly nested loop)

Compute it in parallel

Put scores into a 100-element histogram
Triolet Compiler
Intermediate Representation

- List comprehension and `par` build a package containing
  1. Desired parallelism
  2. Input data structures
  3. Loop body
    for each loop level
- Loop structure and parallelism annotations are **statically known**

```
correlation xs ys =
  let i = IdxNest HintPar
    (arraySlice xs)
    (λx. IdxFlat HintSeq
      (arraySlice ys)
      (λy. f x y )
    )
  in histogram 100 i
```

Outer loop

Inner loop

Body

Joint Lab 2013
Triolet Meta-Library

- Compiler “inlines” list-comprehension into histogram
- Histogram has code paths for handling different loop structures
- Loop structure is known, so compiler can remove unused code paths

```haskell
correlation xs ys =
case IdxNest HintPar
  (arraySlice xs)
  (λx. IdxFlat HintSeq
     (arraySlice ys)
     (λy. f x y ))
  of IdxNest parhint input body.
case parhint
  of HintSeq. code for sequential nested histogram
        HintPar. parReduce input
            (λchunk.
                seqHistogram 100 body chunk)
        IdxFlat parhint input body. code for flat histogram
```

Joint Lab 2013
Example: Correlation Code

- Result is an outer loop specialized for this application
- Process continues for inner loop

```
correlation xs ys =
  parReduce
    (arraySlice xs)
    (\chunk. seqHistogram
      100
      (\x. IdxFlat HintSeq
        (arraySlice ys)
        (\y. f x y)
        chunk)
    chunk)
```
Cluster-Parallel Performance and Scalability

- Triolet delivers large speedup over sequential C
- On par with manually parallelized C for computation-bound code (left)
- Beats similar high-level interfaces on communication-intensive code (right)
Tangram

- A parallel algorithm framework for solving linear recurrence problems
  - Scan, tridiagonal matrix solvers, bidiagonal matrix solvers, recursive filters, ...
  - Many specialized algorithms in literature
- Linear Recurrence - very important for converting sequential algorithms into parallel algorithms
Tangrams Linear Optimizations

• Library operations to simplify application tiling and communication
  – Auto-tuning for each target architecture

• Unified Tiling Space
  – Simple interface for register tiling, scratchpad tiling, and cache tiling
  – Automatic thread fusion as enabler

• Communication Optimization
  – Choice/hybrid of three major types of algorithms
  – Computation vs. communication tradeoff
Linear Recurrence Algorithms and Communication

Brent-Kung Circuit
Ex. Cyclic Reduction

Kogge-Stone Circuit
Parallel Cyclic Reduction

Group Structured
Sectored Thomas
Code Programmers Need to Write: Prefix sum

- **Sequential Code**

  ```c
  SEQ_Compute(...) :
  #pragma unroll
  for(...) :
    UTS_REG(value,i+1) += UTS_REG(value, i);
  ```

- **Tree-Structure Code**

  ```c
  int p=sums[x];
  If (lane_id>=1) sums[tx] = p = p + sums[tx-1];
  If (lane_id>=2) sums[tx] = p = p + sums[tx-2];
  If (lane_id>=4) sums[tx] = p = p + sums[tx-4];
  If (lane_id>=8) sums[tx] = p = p + sums[tx-8];
  If (lane_id>=16) sums[tx] = p = p + sums[tx-16];
  warp_sum[warp_id]=p;
  ```
Tangram Initial Results

Prefix scan on Fermi (C2050)

Prefix scan on Kepler (Titan)

IIR Filter on both GPUs

Tridiagonal solver on both GPUs
Conclusion

- Auto-tuned generic LR algorithms in Tangram outperforms specialized scan, tridiagonal, and IIR algorithms.
- Publish and release Tangram
  - Current tridiagonal solver in CUSPARSE is from UIUC based on the Tangram work
  - Integration with Triolet
- Triolet as an open source project
  - Develop additional Triolet library functions for important application domains
  - Develop Triolet library functions for GPU targets
THANK YOU!