

With great trends come great polyhedral responsibilities

Benoit Meister, Reservoir Labs

time-only

Mat A(1024)(1024)

for i:
 B[i] = A[i] * 2;
 B[i] = x + 3;

S: for i:
 for j:
 for k:

dep: RAR

$d_s = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ → 1. Order // 2. Permutability 3. Locality

$\beta_s = (0, 0, 0, 0)$

$\Gamma_s = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$b_j = \dots$

$G_j[G_j - 2] = \dots$

$ds("true-only");$

High Performance Computing Buzzword Bingo

	Big Data	
Exascale 		Deep Learning 
Heterogeneity		
	Low-Latency	Graph Computing

Opportunity to contribute to a few of the current trends

Lots of fun to be had

Golden era for the polyhedral model

How do we stay golden ?

Outline

Quick context: Reservoir Labs

U.S. Department of Energy Exascale Computing Programs

Context of Reservoir's work

Technical fun

Adoption

Deep Learning optimization

Context of Reservoir's work

Technical fun

Adoption

How could this be even better ?

Reservoir Labs

Compiler R & D

- **R-Stream** polyhedral mapper
- Compiler services

Some joined from polyhedral community

- Benoit Meister (Tech lead)
- Muthu Baskaran
- Tom Henretty

Polyhedral Alumni

- Nicolas Vasilache
- Benoit Pradelle
- Louis-Noel Pouchet
- Cedric Bastoul
- Sanket Tavarageri
- Athanasios Konstantinidis
- Allen Leung, Eric Papenhausen

Many others, from other backgrounds

Other major activities

- **Cybersecurity**
 - R-Scope
- **Tensor-based data analytics**
 - ENSIGN
- **Fast Algorithms**
 - Radar
 - Faster Fourier Transforms

President: Rich Lethin

The R-Stream compiler

Introduced polyhedral engine in 2005

- Version 3.0

Java code

- Plus a few C and C++ bindings

Some strengths:

- Mapping and code gen is driven by a machine model (XML file)
 - Hierarchical, heterogeneous
- Supports hardware features found in most computers
 - Explicit memory management (scratchpads, DMAs)
 - Tiled architectures
- Targets broad set of parallel programming models
 - Annotations, SPMD, runtime APIs
- Has an auto-tuner

DOE Exascale

Exascale at the U.S. Dept. of Energy (DOE)

A bird's eye view

DOE funds basic and applied energy-related research

- High energy physics
- Materials
- Chemistry
- Clean energy
- Biology & Environment



Important areas related to computing:

- Production (Instruments), management and processing of Big Data
- Modeling & **Simulation**
- Cybersecurity

Worked with the polyhedral model on this

But Reservoir is also present & active on these topics

- R-Scope Network Intrusion Detection appliance
- ENSIGN Tensor analytics

Motivation for Exascale

Scientists really have more needs! Exaflops.

- Resolution
 - E.g. can simulate combustion of a cubed-millimeter but not an entire combustion chamber
- Realism
 - Multi-physics, more interrelated PDEs
- Machine learning is permeating DOE research

Not only about who has the bigger machine

Main Challenges with Exascale

All the Petascale challenges, but worse

- Performance
 - Parallelism, locality, load balancing, algorithmic scalability
 - Latency of local & remote memory accesses
- Productivity
 - DSLs, with their flexibility vs performance tradeoff
 - Parallel debugging

Hitting some hardware boundaries

- Process scaling continues
- But energy envelope is bounding HW capabilities

Working around Power Constraints

Lower voltage as much as possible

- Near Threshold Voltage
 - Performance variability across PEs increases
 - Heterogeneity, even in a homogeneous array of PEs

Increase parallelism as much as possible, lower frequency

- Use of hierarchies to get to scale
 - Affects latencies
- Fork-Join, Loop parallelism often not enough to produce that much concurrency

Limit memory bandwidth

Direct impact on software requirements

Parallel programming model must enable

- Fine-grain load balancing
- Non-loop (task) parallelism
 - Even in loop codes
- Hiding long memory latencies

DOE projects widely adopted **Event-Driven Task (EDT)** runtimes

- Declare tasks and their dependences
- Tasks are scheduled asynchronously
 - Work-stealing variants

Reservoir Supported 2 projects with 3 different EDT runtimes:

- Intel: Open Community Runtime (3 versions), CnC
- ET International: SWARM

Many other developments: Legion, HPX, ParSec, etc.

DOE Exascale: Technical Fun

Runnemedede

Intel's Target Exascale Target

A few thousand PEs per chip

One host ("control") processor per 8 PEs

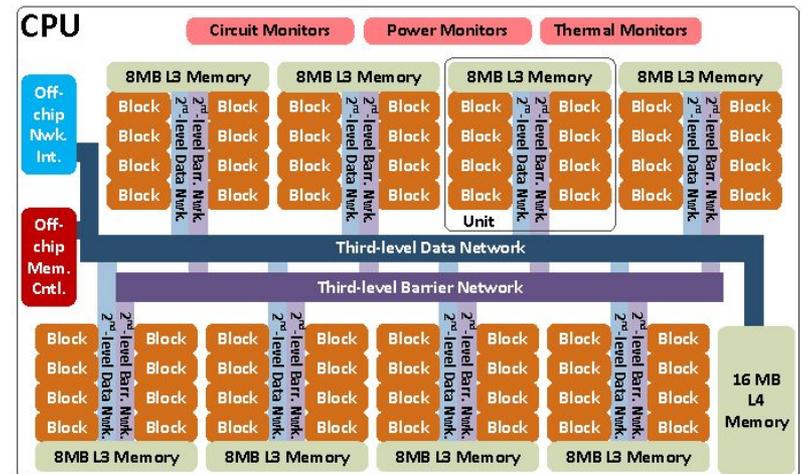
- Dumbed down x86

Non-vector: each PE has its IP

No cache coherency

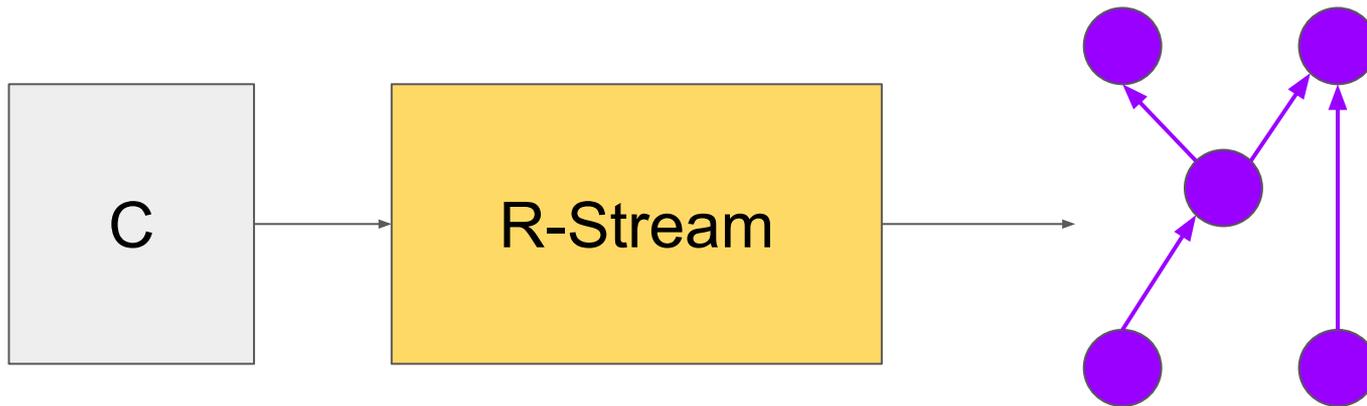
- Scratchpad memory hierarchy
- Optional read-only caches

Near Threshold Voltage



Our contribution

Automatic parallelization of C programs to scalable asynchronous tasks and dependence



Challenges

Producing task parallelism

- Existing literature [Baskaran]
 - Dependence computation didn't scale
 - Tasks need to be carefully scheduled to scale

Explicit data management

- In OCR, data is partitioned into data blocks (DBs)
 - Blocks of contiguous memory
- EDT readiness triggered by two kind of dependences
 - Control
 - Data (a DB)

Scaling task dependence computations

(Problem 1)

Loops have inter-task (outer) and intra-task (inner) dimensions

State of the art [Baskaran]

- Produce a dependence polyhedron
 - Tiled iteration spaces
- Project out intra-task dimensions

Computation of task dependence was too slow

- Tiled dependence polyhedron dimensionality can be high
- Projection is relatively expensive

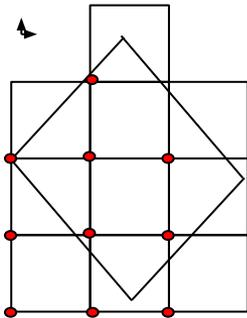
Using pre-tiling iteration spaces (Solution to Problem 1)

Use representation of tiling as a linear relationship

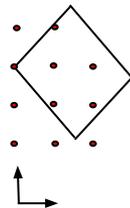
$$I = TJ + K, 0 \leq K < \text{diag}(T)$$

Retain integer J points that correspond to a non-empty task

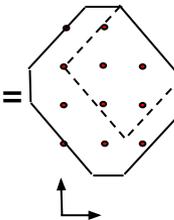
Pre-tiling domain



Naive compression along tiles
Misses non-full tiles!

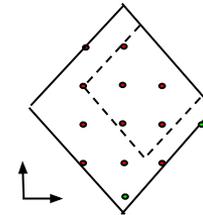


+



Conservative method (P+U)
Includes exact representatives
But more complex shape

Inflation-based method
May include more tile representatives
Same shape as original iteration domain



[Meister]

Representing dependences at runtime

(Problem 2)

We have inter-task dependences in the (source task, dest task) space

- Naive approach: use a synchronization object per dependence
 - $O(n^2)$, impractical even at lower scales
 - Especially if we create them all upfront
- Better approaches use one object per task
 - “Pull” model
 - When done, source task validates task dependence
 - Destination tasks register with all their predecessors
 - Each task maintains the list of its predecessors
 - “Push” model
 - Each destination task knows its # predecessors
 - When done, source task decreases counter for each successor

Limiting runtime task management overhead (Problem 3)

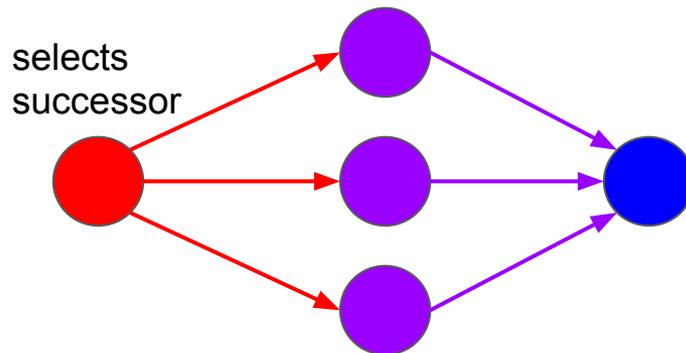
Cost to maintaining a lot of non-ready tasks

Worst case when all tasks need to be created upfront

- Also gets huge Amdahl's law penalty

Best approach: push model with **on-the-fly task creation**

Problem when successor task has >1 predecessors



Decide **who creates the successor task** without introducing extra syncs

In OCR, tasks are atomic: extra syncs means extra tasks (and deps)

On-the-fly task creation

(Solution to Problem 3)

Single node: *first predecessor that is done*

- Decrement successor counter but create it if necessary
 - “Autodec” operation
 - Based on atomics

Multi-node: *agreed upon* predecessor

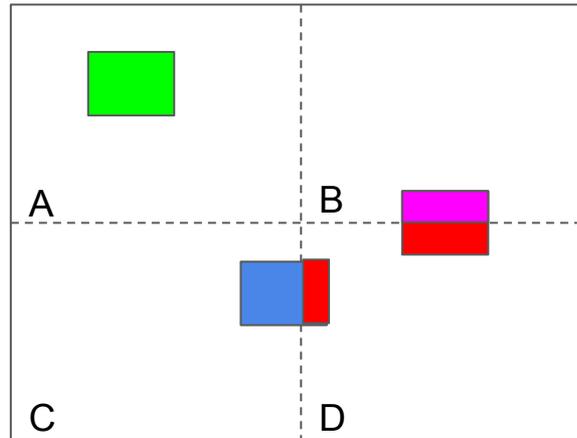
- All predecessors must know it statically **to avoid syncs**
- E.g., lexicographic min of the predecessors
 - But PILP is costly, can produce ugly code
- **Lexico min can be computed at runtime**
 - Early-exited loop
 - Cheap, readable. Yay!

Dealing with data blocks

(Problem 4)

DBs create some challenges

- Introduce index set splitting
 - E.g., some iterations use (DB0, DB1), vs (DB2, DB1)
 - “Static” performance cost



- Read-Written ones create more synchronizations
- Impact on runtime overhead
 - # DBs to manage at any point in time
 - Small DBs: high runtime overhead, less sync

Limiting data block management overhead

(Solutions to problem 4)

Our solution maintains the **#DBs managed to the runtime** low

- Creates a DB for its first user
- Destroys a DB when its last user is done

Solution similar with task management

- Also based on counting

Partial solutions to index set splitting problem

- Can **copy data** from DBs **into local DB** and run without splitting
 - Costs an extra copy. Only worth if
 - Reuse is good
 - Performance benefits greatly from not splitting
- Use map data -> (DB Id, offset within DB) **directly in access functions**
- Cost function is complex.

[Pradelle]

DOE Exascale tools: Adoption

Excellent case for the Polyhedral Model

Programming with tasks, dependences and data blocks is complicated

- Direct calls to API, can be tedious
- Dealing with on-the-fly task creation
- Dealing with data blocks
- Tuning hints

Application writers have to rewrite their code anyway

- Why not just write it in a polyhedral friendly way ?

Excellent case for generating code from a high-level description

However...

We offer a solution for *a portion* of the applications

- Including some sparse codes
- Still not the whole spectrum
- Also R-Stream didn't support Fortran, C++

Application writers still need to code to the runtime for other apps

- Learn the APIs & code with them

Lack of Transparency: what was done to obtain generated code ?

Legacy of overpromising tools

- Application writers won't bother rewriting some of their codes
- Reservoir can do it but the model doesn't scale

Application writers might be uncomfortable with automated competition

- Captious!

Steps taken by Reservoir

Enhancing R-Stream **from parallelizing** compiler **to porting** compiler

- Parallel code as an input
- Deoptimization

Support for **more input languages**

- Through a new “LLVM front-end” to R-Stream
 - Prototype, in good shape but not released yet.

How about the Community ?

- Some efforts at **explaining** automated optimization exist [Bastoul]
- **Increase application domain** of the polyhedral model
 - Dynamically linear codes [Clauss:Apollo]
 - Adaptive mesh computations [Bastoul]

Deep Learning

Context

DARPA PERFECT program

- Teamed up with UC Berkeley around RISC-V RocketChip
 - Tools to specialize hardware
 - Specialized vector processor: Hurricane (1 and 2)
 - Application domain: computer vision
- First used R-Stream to map Caffe Kernels to Hurricane
 - Developed at UCB
- Team moved to Google TensorFlow in 2017
 - Coming out as most popular

Market context

Multiplication of the NN frameworks and representations

- Industry (Google, Intel, Microsoft, Facebook, Baidu, etc.)
- Academia

Multiplication of the Specialized HW

- Google, Nvidia, Intel, ... almost everybody
- Academia

Multiplication of polyhedral solutions ?

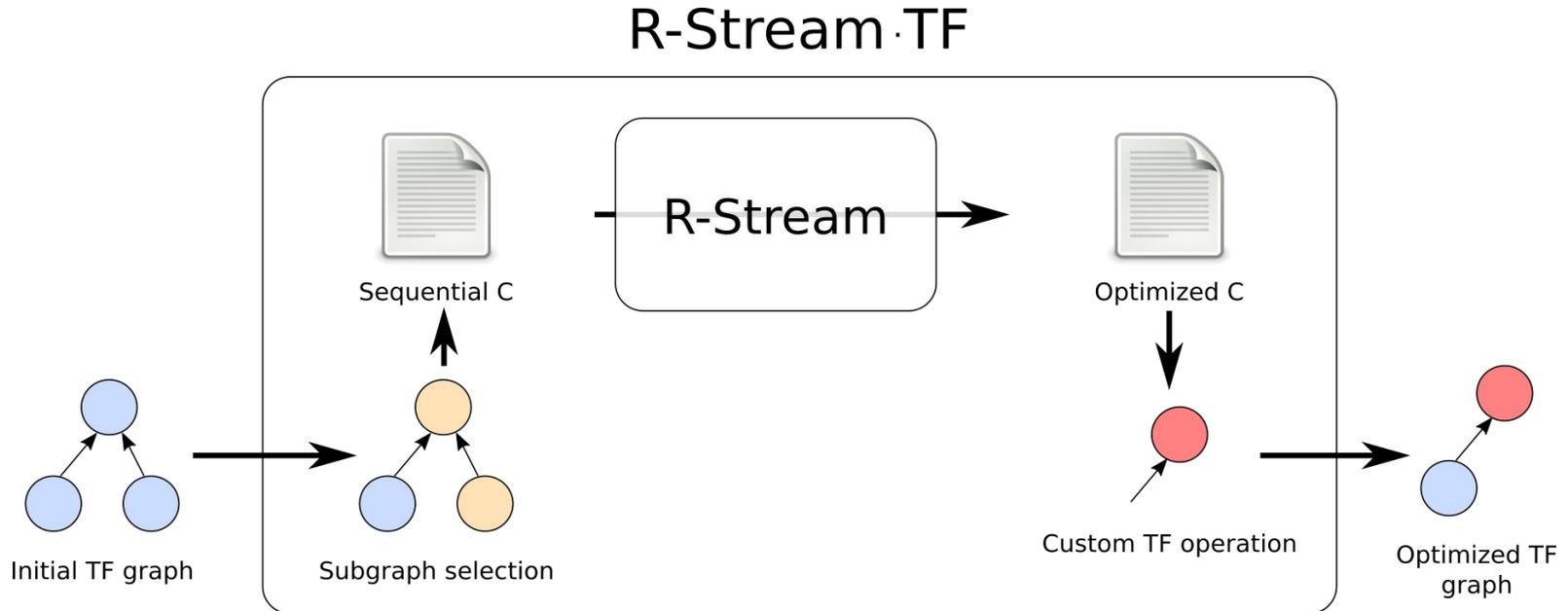
- Reservoir, then Facebook, now Google

Huge stroke of luck

- A lot of the DL kernels have been looked at forever by the community and their tools

Our contribution: R-Stream.TF (tfrcc)

Flow



1. The operator graph is partitioned
2. Sequential C code is generated for every partition
3. R-Stream parallelizes and optimizes the sequential C code
4. The optimized parallel operators are stitched back into the whole graph

High points

Focus is not as much on polyhedral mapper than TF front-end and target backends

Frontend: subgraph formation is tuned to

- Target architecture
- Version of R-Stream used with it

Performance model learned once per (target, rcc) pair

Future work

- Optimize training/gradients
- ... and whatever our customers ask for!

Polyhedral Deep Learning: Adoption

Ecosystem approach

With the DOE, we were seeking adoption by application writers

Here, application writers choose a DL framework

Criteria:

- Hype (esp. for students, dabblers)
- Expressiveness (e.g., RNNs)
- Support for accelerator of choice

Frameworks are a good target for a polyhedral mapper

Most DL Frameworks are open source: we don't *need* their approval

- Better if they collaborate

Ecosystem approach

Some success

- Tensor Comprehensions @Facebook - Insider support
- Google hired a team of polyhedral guys
- Cerebras hired skimo

Would an explosion of Polyhedral DL frameworks be good ?

- Expect it to be limited by small pool of polyhedral experts
- I still see a few “Yet Another Polyhedral Mapper” papers every year, so...
- There will be more

Adoption story is pretty good, here!

Sustainable polyhedral compilation

The polyhedral model enjoys decent adoption in one sector

- How do we maintain it ?
- How do we propagate it to other sectors ?

Consolidate

- Moving R-Stream to LLVM may help
- Find a neutral territory to collaborate
 - Hard problem ?
 - Tool ?

Make it less scary

- Need to make and publish more **user success stories**
- Need to make it easy to teach

Sustainable polyhedral compilation

Resource starvation

Teach the polyhedral model better

When someone asks me for references to learn the model

- I point at several places
 - Slides [Pouchet, Verdoolaege]
 - Ph.D. Theses
- But end up explaining it in a short amount of time

“Everything easy in the polyhedral model has been done” - P. Feautrier

“Polyhedral compilation is easier than you think” - C. Bastoul (approximate)

Thank you!