STREAMING IN THE PGAS ERA

Marco Aldinucci, Maurizio Drocco
University of Torino, Italy
• Programming models

• Distributed Memories: the PGAS model
  • Some example

• Streaming
  • Some example

• Streaming with PGAS
LOW-LEVEL PARALLEL PROGRAMMING MODELS

• Message-Passing
  • Scalability and performance
  • Developer-based precise knowledge of code and overhead
  • Processes + communications (symmetric/collective, blocking/nonblocking)

• Shared-Memory
  • Productivity
  • Global and uniform vision of data layout
  • Threads + synchronisations mechanisms (mutex, atomics, transactions, …)
MPI IS LIKE A CAR, YOU CAN DRIVE DATA WHERE YOU LIKE
–D.K. Panda, leader MVAPICH project at Ohio state Uni.
Val D’Orcia, Tuscany, Italy

EXPECTATIONS
REALITY

Irregular data

Transposed data

Hot data spots
Anyway, MPI is no. 1 in HPC

Courtesy of P. Messina, director of ECP.
No. of software proposals in US ECP 2017
DISTRIBUTED SHARED (VIRTUAL) MEMORY — DSM OR DVSM

- Physically separated memories can be addressed as one logically shared address space
- Hardware or software. Conceptually similar to Virtual Memory
- Designed to distributed platform transparent to programmer i.e. “simplify programming”
- “Vanilla” API
  - `read(addr)`
  - `write(value, adds)`
  - `lock/unlock`
Table A. Software DSM implementations.

<table>
<thead>
<tr>
<th>Implementation</th>
<th>Type of implementation</th>
<th>Type of algorithm</th>
<th>Consistency model</th>
<th>Granularity unit</th>
<th>Coherence policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>IVY</td>
<td>User-level library + OS modification</td>
<td>MRSW</td>
<td>Sequential</td>
<td>1 Kbyte</td>
<td>Invalidate</td>
</tr>
<tr>
<td>Mermaid</td>
<td>User-level library + OS modifications</td>
<td>MRSW</td>
<td>Sequential</td>
<td>1 Kbyte, 8 Kbytes</td>
<td>Invalidate</td>
</tr>
<tr>
<td>Munin</td>
<td>Runtime system + linker + library + preprocessor + OS modifications</td>
<td>Type-specific (SRSW, MRSW MRMW)</td>
<td>Release</td>
<td>Variable size objects</td>
<td>Type-specific (delayed update, invalidate)</td>
</tr>
<tr>
<td>Midway</td>
<td>Runtime system + compiler</td>
<td>MRMW</td>
<td>Entry, release, processor</td>
<td>4 Kbytes</td>
<td>Update</td>
</tr>
<tr>
<td>TreadMarks</td>
<td>User-level</td>
<td>MRMW</td>
<td>Lazy release</td>
<td>4 Kbytes</td>
<td>Update, invalidate</td>
</tr>
<tr>
<td>Blizzard</td>
<td>User-level + OS kernel modification</td>
<td>MRSW</td>
<td>Sequential</td>
<td>32-128 bytes</td>
<td>Invalidate</td>
</tr>
<tr>
<td>Mirage</td>
<td>OS kernel</td>
<td>MRSW</td>
<td>Sequential, Inconsistent, sequential</td>
<td>512 bytes</td>
<td>Invalidate</td>
</tr>
<tr>
<td>Clouds</td>
<td>OS, out of kernel</td>
<td>MRSW</td>
<td>Sequential</td>
<td>8 Kbytes</td>
<td>Discard segment when unlocked</td>
</tr>
<tr>
<td>Linda</td>
<td>Language</td>
<td>MRSW</td>
<td>Sequential</td>
<td>Variable (tuple size)</td>
<td>Implementation-dependent</td>
</tr>
<tr>
<td>Orca</td>
<td>Language</td>
<td>MRSW</td>
<td>Synchronization dependent</td>
<td>Shared data object size</td>
<td>Update</td>
</tr>
</tbody>
</table>

Already mature 20 years ago, now quite rotten

DSM: Why they failed

- Started to simplify distributed code
- To make them efficient, we made the memory consistency model very complex
- This seriously affect the coding effort
**With relaxed consistency ...**

- Can both ifs be evaluated to TRUE?

<table>
<thead>
<tr>
<th>Pi</th>
<th>Pj</th>
<th>A=B=0, B=1, A==0, A=1, B==0</th>
<th>(TRUE, FALSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>write(B,1) if (A==0) ...</td>
<td>write(A,1) if (B==0) ...</td>
<td>A=B=0, B=1, A=1, B==0, A==0</td>
<td>(FALSE, FALSE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A=B=0, B=1, A=1, B==0, A==0</td>
<td>(FALSE, FALSE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A=B=0, A=1, B==0, B=1, A==0</td>
<td>(FALSE, TRUE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A=B=0, A=1, B=1, A==0, B==0</td>
<td>(FALSE, FALSE)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A=B=0, A=1, B=1, B==0, A==0</td>
<td>(FALSE, FALSE)</td>
</tr>
</tbody>
</table>

Ideally NO, under Sequential Consistency NO

Can both ifs be evaluated to TRUE?  Under weaker models, YES
PGAS PROGRAMMING MODEL (DSM EVOLVED)

- A set of processor, each with own local memory
- Part managed as private, part as shared
  - Sharing implemented HW or SW
- Explicitly NUMA
  - Each location has an affinity with a processor
  - Model differentiates between local and remote data partitions
- Explicitly partitioned
  - Collective synchronisations, i.e. barriers and fences
PGAS SYSTEM AT BARE BONES

Partitioned Global Address Space (PGAS)
**APPROACHES**

<table>
<thead>
<tr>
<th>Languages</th>
<th>Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Unified Parallel C (UPC)</td>
<td>• UPC++</td>
</tr>
<tr>
<td>• Co-Array Fortran (CAF)</td>
<td>• OpenSHMEM</td>
</tr>
<tr>
<td>• X10</td>
<td>• Global Arrays</td>
</tr>
<tr>
<td>• Chapel</td>
<td>• DASH</td>
</tr>
<tr>
<td>• STAPL</td>
<td>• ...</td>
</tr>
<tr>
<td>• Titanium</td>
<td></td>
</tr>
</tbody>
</table>
#include <libdash.h>

int main(int argc, char* argv[]) {
    dash::init(&argc, &argv);

    int myid = dash::myid();
    int size = dash::size();

    dash::array<int> key(size);

    if (myid==0) {
        for (i=0; i<size; i++) key[i]=compute_key(...);
    }

    dash::barrier();

    cout<<"Hello from unit "<<myid<<" of "
        <<size<<" my key is "<<key[myid]<<endl;

    dash::finalize();
}
OpenSHMEM “symmetric” memory model

formerly known as Cray SHMEM (1993)
**OPENSHMEM**

**A OLD FASHIONED PGAS**

- Perform computations in separate address spaces and explicitly pass data to and from different processes in the program

- **API**
  - Library Setup: init, query
  - **Symmetric** Data Object Management: allocation, deallocation, reallocation:
  - Remote Memory Access: get (private ← shared), put (shared ← private)
  - **Atomics**: swap, inc, add, was, fetch_add
  - **Synchronisation** and Ordering: fence (i.e. flush inflight ops), quiet (ensure other completion), barrier
  - Collective Communication: broadcast, collection (i.e. gather), reduction
  - Mutual Exclusion: lock, testlock, unlock
OPENSHMEM HELLO WORLD!

#include <stdio.h>
#include <mpp/shmem.h>
int main (int argc, char **argv) {

    int me, npes;

    start_pes (0); /*Library Initialization*/
    me = _my_pe ()
    npes = _num_pes ();
    printf ("Hello World from node %d of %d\n", me, npes);
    return 0;
}
#include <stdio.h>
#include <shmem.h>

long pSync[_SHMEM_BARRIER_SYNC_SIZE];
int x = 10101;

int main(void)
{
    int i, me, npes;

    for (i = 0; i < _SHMEM_BARRIER_SYNC_SIZE; i += 1){
        pSync[i] = _SHMEM_SYNC_VALUE;
    }

    shmem_init();
    me = shmem_my_pe();
    npes = shmem_n_pes();

    if(me % 2 == 0){
        x = 1000 + me;
        /*put to next even PE in a circular fashion*/
        shmem_int_p(&x, 1, (me+2)%npes);
        /*synchronize all even pes*/
        shmem_barrier(0, 1, (npes/2 + npes%2), pSync);
    }
    printf("%d: x = %d\n", me, x);
    return 0;
}
class Node {
    var data: real;
    var next: Node;
}

// List init (seq, with remote operations)
var head = new Node(0);
var current = head;
for i in 1..numLocales-1 do
    on Locales[i] {
        current.next = new Node(i);
        current = current.next;
    }

// List walk (seq)
current = head;
while current {
    writeln("node with data = ", current.data, " on locale ", current.locale.id);
    current = current.next;
}
writeln();

// Data-driven List walk (work done in parallel
// Each locale WRITE its own data (own-compute)
current = head;
while current {
    on current {
        writeln("node with data = ", current.data, " on locale ", here.id);
        current = current.next;
    }
}

// Deallocate (seq)
current = head;
while current {
    on current {
        var ptr = current;
        current = current.next;
        delete ptr;
    }
}
writeln();
PGAS

• Overcomes some DSM criticality

• Aiming at productivity for large scale
  • To overcome MPI — not succeeding yet
  • Most of them build on top or interoperate with MPI

• Designed for weak scalability (own-compute) → Data Parallelism
  • Not for streaming
Coroutines: Semantics in Search of a Syntax

by M. Douglas McIlroy

Oxford University and
Bell Telephone Laboratories, Incorporated.

Abstract: Unlike subroutines, coroutines may be connected, and reconnected, in nonhierarchical arrangements. Coroutines are particularly useful for generating and processing data streams. Semantics for coroutines are developed and examples are given.

Streams! Coroutines: McIlroy, 1968
Symposium on Theoretical Programming Novosibirsk – 1972
IEEE STC Dataflow and Beyond
HTTP://DFSTC.CAPSL.UDEL.EDU

STC Founding Member

These are the people that made possible the creation of this IEEE Special Technical Community.

- Arvind
  Massachusetts Institute of Technology, USA
- David Abramson
  The University of Queensland, Australia
- Georgi Gaydadiev
  Massey, Imperial College London, UK
- Guang R. Gao
  University of Delaware, USA
- Hiranki Kei
  The University of Tokyo, Japan
- Hironori Kasahara
  Waseda University, Japan
- Jack B. Dennis
  Massachusetts Institute of Technology, USA
- Jean-Luc Gaudiot
  University of California, Irvine, USA
- Jin Hai
  Harbin Institute of Technology, China
- Kuan-Ching Li
  Providence University, Taiwan
- Mateo Valero
  Universitat Politecnica de Catalunya, Spain
- Mei Hong
  Beijing Institute of Technology, China
- Nahid Emad
  University of Versailles, France
- Nelson Amaral
  University of Alberto, Canada
- R. Govindarajan
  Indian Institute of Science, India
- Roberto Giorgi
  Universita di Salerno, Italy
- Skevos Evripidou
  University of Cyprus, Cyprus
- Stephanie Zuckerman
  Michigan Technological University, USA
- Vivek Sarkar
  Rice University, USA
- Won Woo Ro
  Yonsei University, South Korea
- Zheng Weimin
  Tsinghua University, China

Members of our STC

- Dr. Mario Donato Marino
- Dr. Lorenzo Verdoscia
- Dr. Marco Procaccini
- Dr. Chen Liu
- Jin Yan
- Tongsheng Geng
- Dr. Sreepathi Pai
- Peng Qu
- Dr. Albert Cohen
- Shipeng Qi
- Dr. Toshiaki Kitamura
- Jose M Monsalve Diaz
- Dr. Erik Altman
- Dr. Karu Sankaralingam
- Dr. Jidong Zhai
- Professor Eduardo Juarez
- Dr. Andres Marquez
- Yoshitake Oki
- Professor Junqing Yu
- Professor Wenguang Chen
- Professor Sunita Chandrasekaran
- Professor Dongrui Fan
- Joshua Sutterlein
- Professor Bob Iannucci
- Dr. Long Zheng
- Dr. Hao Tu
- Dr. Tobias Becker
- Siddhansetk Raskar
- Professor Marco Aldinucci
- Professor Avi Mendelson
- Professor Xiaoming Li
- Dr. Xu Tan
• Data Stream Processing (DaSP)

• Real-time processing of continuous data streams

• Processed on-the-fly with stringent Quality of Service (QoS)

• Potentially irregular flows of data must be timely processed
  • detect anomalies, real-time incremental responses, etc.
Stream processing applications

INTERSECTING DIFFERENT CS AREAS

• Control Systems (and CPS)
  • Including network processors, FPGAs, etc

• Parallel Computing (shared memory)
  • StreamIt (2006), TBB, FastFlow (2009), RaftLib (2013), StreamBox (2016), …

• BigData Analytics
  • Lambda, Kappa architectures
  • Apache + \( x \in \{Kafka, Flink, Storm, Apex, Spark, \ldots \text{ and counting}\} \)
  • Tensorflow (2015)
KEEP CALM AND INGEST (WITH CALM) THE STREAM
STREAMING: CORE PARADIGM

control actors

operator

T-gate

F-gate

Boolean actors

V

or

decider

T F

merge

and

Figure 2. Node types for data flow programs.
NETWORK OF EXECUTORS VS TASK GRAPH

• Graph of tasks
  • Dataflow — typically DAG
  • Each node is a task
  • E.g. a C++ object
  • Problems: firing, scheduling, etc.

• Network of executors
  • "Controlflow" — typically cyclic graph
  • E.g. threads or processes
  • Problems: pinning, mapping, pooling, etc.
FastFlow
http://mc-fastflow.sourceforge.net/

• **Toreador** (EC-RIA, H2020, ICT-16-2015 big data): TrustwOrthy model-awaRE Analytics Data platfORm (2016, 36 months, total cost 6.5M €)


• **REPARA** (EC-STREP, 7th FP): Reengineering and Enabling Performance And poweR of Applications (2013, 36 months, total cost 3.5M €)

• **ParaPhrase** (EC-STREP, 7th FP): Parallel Patterns for Adaptive Heterogeneous Multicore Systems (2011, 42 months, total cost 4.2M €)

• **IBM Research** 3 faculty awards 2015 (50K $)

• **Noesis Solutions**: Machine learning for engineering 2015 (75K €)

• **A3CUBE Inc.**: FastFlow/PGAS with in memory fabric 2014

• **NVidia Corp**: CUDA Research Center at University of Torino 2013
SWSR QUEUES + MEDIATORS

- **FF bound shmem FIFO channel**: Single-Producer-Single-Consumer lock-free fence-free queue
- **FF unbound shmem FIFO channel**: Single-Producer-Single-Consumer lock-free fence-free queue
- **FF (lock-free) distributed memory channel**: RDMA or TCP

### Graphs

- **GPU node**: channel name or channel
- **Node**: channel name or channel
- **mi-node**: channel names
- **mo-node**: channel names
- **Process**: network symmetric or asymmetric (scatter, gather, etc)
- **Distributed node**: distributed

**Notes**:
- GPU bound shmem FIFO channel
- Single-Producer-Single-Consumer
- lock-free fence-free queue
- GPU unbound shmem FIFO channel
- Single-Producer-Single-Consumer
- lock-free fence-free queue
- GPU (lock-free) distributed memory channel
- RDMA or TCP
**Generate the Network**

**True data dependencies moves across arrows**

Composing via mediator guarantee correctness (data races & deadlock freedom)

Possible data race
PROGRAMMING MODEL: SYNCHRONISATIONS HAPPEN BY WAY OF P2P DATA DEPENDENCIES (THUS NO ATOMICS ARE NEEDED)

Shared-memory cache-coherent or non-coherent multicore

Distributed GAM

Synchronisations are in a message-passing style, but designers are not forced to think in a distributed way

No copies are needed, the memory fences are but asynchrony helps

One-sided
either eager “put” or lazy “get”
Based on Single-Writer-Single-Reader FIFO

- Does not require atomic
  - No fence under TSO, WriteFence under WO
    - J. Giacomoni et al. Fastforward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue. PPoPP 08
- Enough to support Producer-Consumer
  - Inherently asynchronous
  - Powerful enough to build a general purpose parallel programming model

FastFlow unbound queue
core-to-core message latency
Xeon E7-4820 @2.0GHz Sandy Bridge
Bringing Parallel Patterns out of the corner: the P³ARSEC Benchmark Suite

**A Benchmark Suite** for parallel patterns-based applications
**Parallel Pattern** design of 12 out of 13 PARSEC benchmark applications
Implementations with FastFlow and SkePU2 publicly available

Comparison over 3 different shared memory multicore architectures (Intel Xeon, Intel Xeon Phi, IBM Power 8) using different implementations

- **Execution Time**
- **Lines of Code**

*Normalized with respect to Pthreads, averaged over all the benchmarks (the lower the better)

**Detailed results** and comparison with additional frameworks can be found in:

**FASTFLOW**

http://calvados.di.unipi.it/paragroup/

- Compared in performance and features against: openMP, TBB, Pthreads, OpenSs, MPI, ...
- 80+ papers from 2010
Windowed Stream Processing
**Windowed Stream Processing**

- Windows approximate infinite stream history
  - tuple significance is often time-decaying
  - only the most recent tuples are kept
- Different windowing policies
  - Sliding windows: window size + sliding factor
  - Session windows [Apache Flink, Apache Beam…]
- Common implementation = Worker-side windowing (more parallelism)
KEY-PARTITIONING (KP)

\[\text{IN}(A) \quad \text{IN}(B)\]

\[S \quad A \quad B\]
WINDOW-FARMING (WF)

IN(A) = IN

IN(B) = IN

S -> A

S -> B
A NEW PROPOSAL FOR A STREAM-ORIENTED PGAS
DISTRIBUTED FASTFLOW v.2 — C++ GLOBAL MEMORY STACK

GAM Nets  Task RTS  ...
Smart Global Pointers
GAM
Communication Library
libfabric, MPI...
Platform


Aldinucci, S. Campa, M. Danelutto, P. Kilpatrick, and M. Torquati, “Targeting Distributed Systems in FastFlow,” in Euro-Par 2012 Workshops,
GLOBAL ASYNCHRONOUS MEMORY
A NEW PROPOSAL FOR A STREAM-ORIENTED PGAS (BASED ON FF)

• From MPI Style:
  communicate pointers
  (a.k.a., capabilities)

• From DSM Style:
  shared address space

→ Capability = both data reference and synchronization token
PUBLIC CAPABILITIES

- Read-only (single assignment)
- Cacheable
- Can be copied
PRIVATE CAPABILITIES

• Exclusive read-write
• Not cacheable
• Can be moved
(Trivial) Sequential Consistency

Avoiding consistency issues (vs solving as in DSM/PGAS)

SWMR cache-coherence invariant:
  - public → (NW)MR
  - private → SWSR
SMART GLOBAL POINTERS

• Rooted in modern C++
  • Intentional programming:
    public → shared, private → unique

• Automatic Memory Management — the C++ way
  • Smartness = memory-reference lifetime binding
  • No memory leaks, no dangling pointers
  • No garbage collection (vs Java & friends)
**PUBLIC POINTERS**

```cpp
public_ptr(T * const, Deleter);
public_ptr<T> make_public(Args&&...);

// copy constructor/assignment...
// move constructor/assignment...

public_ptr(private_ptr<T> &&);
public_ptr& operator=(private_ptr<T> &&);

std::shared_ptr<T> local();

void push(executor_id to);
public_ptr<T> pull_public(const exec_id from);
public_ptr<T> pull_public();
```

enables plain C++ code
PRIVATE POINTERS

private_ptr(T * const, Deleter);
private_ptr<T> make_private(Args&&...);

//move constructor/assignment...

//NO copy constructor/assignment

gam_unique_ptr<T> local(); //unique_ptr + custom deleter

void push(executor_id to);
private_ptr<T> pull_private(const exec_id from);
private_ptr<T> pull_private();

enables plain C++ code
SMARTNESS FOR PUBLIC POINTERS

- Distributed **reference counting** protocol
- Creation/copy/push trigger +1, destruction triggers -1
- C++ shared pointers: atomic-based reference counting
SMARTNESS FOR PRIVATE POINTERS

- Distributed memory releasing protocol
- Destruction triggers releasing
- C++ unique pointers: destruction-triggered release
- Inherently simpler than public pointers (as unique vs shared)
KP - Inherently Efficient

- Disjoint IN(A)/IN(B)
  - each tuple accessed “exclusively”

- GAM implementation
  - private pointers - exclusive capabilities
  - 1 RMA access per tuple
WF - Inherently Complex

- \( IN(A) = IN(B) = IN \)
  - each tuple accessed “by-any”

- GAM implementation
  - public pointers - read-only replicas
  - multiple RMA accesses per tuple
GAM ADVANTAGES

• Passing capabilities versus data
  • efficient worker-side windowing
  • extreme case: static dispatching not viable
HTP Video Restoration

Original Baboon 1024x1024

10% impulsive noise

50% impulsive noise

90% impulsive noise

Restored
HTP Video Restoration

- A GAM implementation of ordering farm

GAM node

GAM communicator

C++/GPU code [GTC 2014]
HTP Video Restoration

720p Video Stream

Throughput (fps)

<table>
<thead>
<tr>
<th>Noise</th>
<th>GAM+GPU</th>
<th>GPU [GTC 2014]</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
<td>220</td>
<td>55</td>
</tr>
<tr>
<td>70%</td>
<td>165</td>
<td>110</td>
</tr>
</tbody>
</table>

OCCAM cluster @ UniTO
4x GPU nodes
Nvidia Tesla k40 GPUs
EuroPar 2018
Torino, Italy — 27-31 August 2018

Co-chairs: M. Aldinucci, L. Padovani, M. Torquati