Composition and Compartmentalisation As Enabling Features for Data-Centric, Extreme Scale Applications: An MPI-X Approach

Marco Aldinucci, Claudia Misale, Maurizio Drocco - University of Torino, Computer Science Department
Massimo Torquati - University of Pisa, Computer Science Department
for Scientific Computing

April 12-15, 2016
The farm pattern. Composed in a pipeline pattern.

Patterns defines collective behaviours, as MPI collectives, they are just more abstract

Mediator nodes enforce compositionality and compartmentalisation
Thank you!
Questions?
Task-based programming models
minisymposium @Siam-PP 2016 - Outline

✤ Computations described by patterns
  ✤ Already mainstream in several areas, e.g. data analytics, GPUs, …

✤ Language: C++11 - Ideally nothing more
  ✤ Streams and data parallelism is already in the syntax
  ✤ Tasks can be described with std::functions, C++11 lambdas, C++11 attributes …

✤ Run-time is my specific interest
  ✤ FastFlow - based on hybrid actor/CSP model - originally for sh-mem moving to large scale
  ✤ Does we need MPI+X or MPI-X?
FastFlow (FF) and its data-centric run-time support

- Core patterns: pipeline, farm, feedback
- High-level patterns: `parallel_for`, `parallel_forReduce`, ...
- Building blocks: queues, `ff_node`, ...
- Parallel applications: efficient and portable

Building blocks:
- CUDA
- OpenCL
- TCP/IP
- IB/OFED

Multicore and many-core platforms:
Clusters of multicore + many-core
Building blocks

non-blocking threads
(can switch to blocking at
runtime by way of a
native protocol)

nonblocking thread
GPU/accelerators
(OpenCL, CUDA)

process of nonblocking threads
distributed zero-copy
TCP/IP, OFED/IB, MPI (ongoing),
HW/SW PGAS (ongoing)
Building blocks
Everything is SPSC (in the shared-memory)

- Enough to support producer-consumer
  - Inherently weaker w.r.t. mutual exclusion
  - Weaker execution model requirements
    - Mutex not really possible on SIMT model (GPU)
    - Mutex requires memory-fences and leverages on (expensive) cache coherency on multicore
  - Deadlock is cyclic networks avoided via unbound queue (wait-free)

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

Distributed zero-copy channel
0MQ/TCP or native IB/OFED

shmem channels communicate
pointers in a message passing style
FastFlow unbound shared-men wait-free queue (EuroPar 2012)

Basically a ring-buffer of ring-buffers equipped with a lock-free memory allocator

FastFlow distributed-mem lock-free queue (unpublished)

- Lock-free ring buffer over HW-supported non coherent distributed shared memory
  - Cray Gemini, A3Cube Inc. Ronniee Express, (Omnipath?)
  - HP: remote write, local read only. Duplication of indexes with programmatic control of (lazy) consistency
- Enough to implement the whole libfabric interface
  - Ultra-stable: latency < 800 ns at 99 percentile
  - Asynchronous
FastFlow data movement model
hybrid shared-memory & message passing

Shared-memory cache-coherent multicore

Distributed PGAS or non coherent shared-memory
Core patterns: farm, pipeline, feedback

+ Nonblocking/blocking, dynamic/static scheduling, E/C policy configuration, core pinning

Patterns

farm

master worker

farm with feedback

farm with in-memory scheduling

farm of GPU nodes

pipeline

pipeline with feedback

pipeline with GPU nodes

Specialisation (OO)
Core patterns: farm, pipeline, feedback
+ Nonblocking/blocking, dynamic/static scheduling, E/C policy configuration, core pinning

Patterns

farm

```
E
W1
W2
... 
Wn
```

```
S1
S2
... 
Sn
```

pipeline

```
master
worker
farm with
feedback
farm with
in-memory
scheduling
farm of GPU
nodes
```

```
E
W1
W2
... 
Wn
```

```
S1
S2
... 
Sn
```

Specialisation (OO)

pipeline with feedback

pipeline with GPU nodes

Mem lockfree

GPU-1

GPU-n
Stream: Bowtie (BT) and BWA sequence alignment tools

- Top tools for parallel DNA alignment
- Hand-tuned C/C++/SSE2 code
- Spin locks + Pthreads
- Sequences (reads) are streamed from MMIO files to workers
- Memory bound

- FastFlow master-worker
- Memory affinity, pinning, affinity scheduling (embedded in the pattern)
- BT: up to 200% speedup
- BWA: up to 10% speedup over originals

Macro-Data-Flow (MDF) pattern encoding dependence DAG

- pipeline(TaskGen, farm(TaskExec))
- Configurable scheduling, affinity, …

Dynamic generation of the DAG

Comparable or faster than specialised linear algebra frameworks (e.g. PLASMA)

MDF is general, can be used for a wide range of applications, Dynamic Programming, Divide&Conquer, …
Core patterns can be arbitrarily composed

pipe(farm(s), farm(s))

farm(pipe(s1, s2))

farm(farm(s))

pipe(farm(s), farm(s), loopback)

pipe(s1, farm(s2))
Core patterns: a programming model

- A low/medium-level data-centric programming model
  - Concurrent computation modelled as a (cyclic) graph
  - Nodes are parallel activities. Edges are true data dependencies
  - Synchronisation are messages, data can be moved as messages or shared memory
    - Can be realised with or without coherency, in shared-memory, distributed, PGAS, …

- A CSP-actor hybrid model
  - Processes are named and the data paths between processes are identified
  - Recently started also to look at IoT
High-level patterns - address application needs

✦ Loop parallelism (OpenMP fashion)
 ✦ Parallel-For, Parallel-For-Reduce
 ✦ PDP 2014 - Danelutto et al. Loop parallelism: a new skeleton perspective on data parallel patterns

✦ Data Parallelism
 ✦ Stencil, Stencil-Reduce, Map, MapReduce (pipelined)
 ✦ J. Supercomputing 2016. Aldinucci et al. - The Loop-of-Stencil-Reduce paradigm

✦ Task & Stream
 ✦ Pool (e.g. genetic alg.), Macro-Data-Flow (e.g. linear algebra, dynamic programming, …), Tasks
 ✦ Farm, Pipeline
 ✦ PPoPP 2016 - De Matteis et al. - Keep Calm and React with Foresight: Strategies for Low-Latency and Energy-Efficient Elastic Data Stream Processing

✦ Implementation-wise, just OO extensions of composition of core patterns
Example: parallel for

Currently a method call on a C++11 Lambda function (loop body)
- All other high-level patterns in the same style

Or with C++11 generalised attribute syntax (N2761)
- Within REPARA EU-FP7 STREP project

```cpp
// FastFlow (--std=c++11)
ff::ParallelFor pf;
pf.parallel_for(0L,N,[&A](const long i) {
    A[i]++;
},nworkers);

// OpenMP (-fopenmp)
#pragma omp parallel for num_threads(nworkers)
for(long i=0;i<N;++i) {
    A[i]++;
}
```

```cpp
[[ff::target(ff::cpu,ff::gpu), ff::input(A), ff::output(A), ...]]
for( ; ; ; ) { ... }
```
Example: independent tasks, (but introducing true-dependencies is easy)

```cpp
#include <ff/taskf.hpp>
#include <ff/parallel_for.hpp>
using namespace ff;

void F(long *E) {
    printf("Executing %ld\n", *E);
    delete E;
}

int main(int argc, char *argv[]) {
    int W = ff_numCores();
    if (argc>1) W = atoi(argv[1]);
    ff_taskf taskf(W);

    // start the scheduler and workers
    taskf.run();
    taskf.AddTask(F, new long(1));
    taskf.AddTask(F, new long(2));
    taskf.AddTask(F, new long(3));
    taskf.wait(); // barrier

    // add another task,
    // scheduler is stopped
    taskf.AddTask(F, new long(4));
    taskf.AddTask(F, new long(5));

    // run the scheduler
    // with 1 thread and then barrier
    taskf.run_then_freeze(1);

    // here the scheduler is stopped
    taskf.AddTask(F, new long(6));
    taskf.AddTask(F, new long(7));
    taskf.AddTask(F, new long(8));

    // run the scheduler then barrier
    taskf.run_then_freeze(2);

    // here the scheduler is stopped
    taskf.AddTask(F, new long(9));
    taskf.AddTask(F, new long(10));

    taskf.run_then_freeze();
    ...
```
Example: Distributed Map-reduce with C++ DLambda (SkeDaTo)

```cpp
int main() {
    ...
    skedato::DLambdaInMap<myPair> mapL(
        [] (myPair x) {
            return myPair(x.a, x.a + x.b);
        });
    skedato::init(); //SPMD-free from here
    skedato::DPC<myPair> dpc(LEN);
    ...
    dpc.map_inplace(mapL);
    ...
}
```

### Table I
Performance of dot-product on the cluster.

<table>
<thead>
<tr>
<th>2^{30} double/int DPC collection (4+8 = 12 GB)</th>
<th>sequential implementation</th>
<th>SkeDaTo implementation</th>
<th>native MPI implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>nodes</td>
<td>init (ms)</td>
<td>zip (ms)</td>
<td>reduce (ms)</td>
</tr>
<tr>
<td>-------</td>
<td>-----------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>1</td>
<td>6202.60</td>
<td>3102.12</td>
<td>1688.31</td>
</tr>
<tr>
<td>2</td>
<td>3474.70</td>
<td>1642.60</td>
<td>962.26</td>
</tr>
</tbody>
</table>
Example: Loop-of-stencil-reduce & Pluto

1 std::string stencilf = ff.stencilKernel2D_OCL(
2     "unsigned char", "in", //element type and input
3     "N", "M", //rows and columns
4     "i", "j", "i-", "j-", //row-column global and local indexes
5 std::string("\n") +
6     /* begin of the OpenCL kernel code */
7     "unsigned char n_alive = 0;\n" +
8     "n_alive += i>0 && j>0 ? in[i-1][j-1] : 0;\n" +
9     "... +
10     "n_alive += i<N-1 && j<M-1 ? in[i+1][j+1] : 0;\n" +
11     "return (n_alive == 3 || (in[i][j] && n_alive == 2));"
12     /* end OpenCL code */;
13
14 std::string reducesf = ff.reduceKernel_OCL(
15     "unsigned char", "x", "y", "return x + y;");
16
17 ff::ff_stencilReduceLoop2DOCL<goTask> goSLR(
18     stencilf, reducesf, 0, iterf, // building blocks
19     N, N, NACC, // matrix size and no. of accelerators
20     3, 3); // halo size on the 2 dimensions

Fig. 1: Implementation of Game of Life [13] on top of the Loop-of-stencil-reduce API in FastFlow.
“Exascale programming model will be largely based on MPI+X, where X might be OpenMP, OpenCL, OpenACC … ”

MPI

is like a car, you can drive it as you like
MPI, threads, HW-CAS spinlocks, … are like cars, you can drive it as you like
#include <stdio.h>
#include "mpi.h"

#define MAXPROC 8    /* Max number of processes */
#define NAMELEN 80   /* Max length of machine name */
#define LENGTH 24    /* Length of send buffer is divisible by 2, 4, 6 and 8 */

main(int argc, char* argv[]) {
    int i, j, np, me;
    const int nametag  = 42;    /* Tag value for sending name */
    const int datatag  = 43;    /* Tag value for sending data */
    const int root = 0;         /* Root process in scatter */
    MPI_Status status;          /* Status object for receive */

    char myname[NAMELEN];             /* Local host name string */
    char hostname[MAXPROC][NAMELEN];  /* Received host names */

    int x[LENGTH];        /* Send buffer */
    int y[LENGTH];        /* Receive buffer */

    MPI_Init(&argc, &argv);                /* Initialize MPI */
    MPI_Comm_size(MPI_COMM_WORLD, &np);    /* Get nr of processes */
    MPI_Comm_rank(MPI_COMM_WORLD, &me);    /* Get own identifier */

    gethostname(&myname, NAMELEN);    /* Get host name */

    if (me == 0) {    /* Process 0 does this */
        /* Initialize the array x with values 0 .. LENGTH-1 */
        for (i=0; i<LENGTH; i++) {
            x[i] = i;
        }

        /* Check that we have an even number of processes and at most MAXPROC */
        if (np>MAXPROC || np%2 != 0) {
            printf("You have to use an even np (at most %d)
", MAXPROC);
            MPI_Finalize();
            exit(0);
        }

        printf("P%d on host %s is distributing array x to all %d processes

", me, myname, np);

        /* Scatter the array x to all processes, place it in y */
        MPI_Scatter(&x, LENGTH/np, MPI_INT, &y, LENGTH/np, MPI_INT, root, 
                     MPI_COMM_WORLD);

        /* Print out own portion of the scattered array */
        printf("Process P%d on host %s has elements", me, myname);
        for (i=0; i<LENGTH/np; i++) {
            printf(" %d", y[i]);
        }
        printf("\n");
    }
    else { /* all other processes do this */
        /* Check sanity of the user */
        if (np>MAXPROC || np%2 != 0) {
            MPI_Finalize();
            exit(0);
        }

        /* Receive the scattered array from process 0, place it in array y */
        MPI_Scatter(&x, LENGTH/np, MPI_INT, &y, LENGTH/np, MPI_INT, root, 
                     MPI_COMM_WORLD);

        /* Send own name back to process 0 */
        MPI_Send (&myname, NAMELEN, MPI_CHAR, 0, nametag, MPI_COMM_WORLD); /* Send the received array back to process 0 */
        MPI_Send (&y, LENGTH/np, MPI_INT, 0, datatag, MPI_COMM_WORLD);
    }

    MPI_Finalize();
    exit(0);
}

/* Receive messages with hostname and the scattered data */
/* from all other processes */
for (i=1; i<np; i++) {
    MPI_Recv (&hostname[i], NAMELEN, MPI_CHAR, i, nametag, 
               MPI_COMM_WORLD, &status);
    printf("Process P%d on host %s has elements", i, hostname[i]);
    for (j=0; j<LENGTH/np; j++) {
        printf(" %d", y[j]);
    }
    printf("\n");
}

printf("Ready\n");
Message-passing (e.g. MPI)
Shared-memory (e.g. threads + mutex/CAS)

- (Can be) Efficient
- Available in almost all platforms
  - Often the only access to net, e.g. HRLS Cray XE6 Hermit: 32 x 3552 nodes = 113 664 cores
  - De-facto standards, used for decades

- Parallel primitives fully inter-waived with business code
- How compose “computing phases”, “SW modules” …
  - Parallel behaviour and data layout not explicit in the code. Primitives often do not compose.
- Often to be coupled with shared-memory for intra-node
  - e.g. MPI & Pthreads, MPI & OpenMP, PGAS-MPI & OpenCL, …
Message-passing (e.g. MPI)
Shared-memory (e.g. threads + mutex/CAS)

- (Can be) Efficient
- Available in almost all platforms
  - Often the only access to net, e.g. HRLS Cray XE6 Hermit: 32 x 3552 nodes = 113,664 cores
  - De-facto standards, used for decades

- Parallel primitives fully inter-waived with business code
- How compose “computing phases”, “SW modules” ...
  - Parallel behaviour and data layout not explicit in the code. Primitives often do not compose.
- Often to be coupled with shared-memory for intra-node
  - e.g. MPI & Pthreads, MPI & OpenMP, PGAS-MPI & OpenCL, ...
Message-passing (e.g. MPI)
Shared-memory (e.g. threads + mutex/CAS)

- (Can be) Efficient
- Available in almost all platforms
  - Often the only access to net, e.g. HRLS Cray XE6 Hermit: 32 x 3552 nodes = 113 664 cores
  - De-facto standards, used for decades
- Parallel primitives fully inter-waived with business code
- How compose “computing phases”, “SW modules” …
  - Parallel behaviour and data layout not explicit in the code. Primitives often do not compose.
- Often to be coupled with shared-memory for intra-node
  - e.g. MPI & Pthreads, MPI & OpenMP, PGAS-MPI & OpenCL, …
Val D’Orcia, Tuscany, Italy

Designer utopia
False sharing

Reality

Irregular data

Hot spots
Exascale-like architecture?

- 4-5 levels with different
  - execution model
  - latency, bandwidth
  - synchronisation model
  - data movement model
  - ...

((MPI+X)+X)+ ... +X)
Scientific computing vs analytics

- Scientific simulations
- Tensors with regular, static stencils
- Analytics tens
- Tensors with irregular stencils
- Graphs ...
MPI, the current dominant programming model for parallel scientific programming, forces coders to be aware of the exact mapping of computational tasks to processors. This style has been recognised for years to increase the cognitive load on programmers, and has persisted primarily because it is expressive and delivers the best performance. [Snir et al 1998] [Gursoy and Kale 2004]

Because we anticipate a massive increase in exploitable concurrency, we believe that this model will break down in the near future, as programmers have to explicitly deal with decomposing data, mapping tasks, and performing synchronisation over thousands of processing elements. [Asanovic et al 2006]
Programming model: my wish list

- Should enforce to think to problems in parallel & and at high-level of abstraction
  - Clear semantics: functional and extra-functional (parallel), describing collective behaviours
  - Trading memory coherency for power, and power for parallelism should be a matter of the implementation

- Should support containment and composition
  - At large scale: clear fault model with containment & recovery

- Should integrate synchronisation/communication with scheduling
  - Weak execution model rather than per device. Multicore, GPGPUs, distributed with an unifying vision
Assessment

- MPI more useful as communication library than a programming model
  - MPI-SPMD to start. Exascale code is not one code. It should be a collection of composable and compartmentalised modules.
- Mediator nodes helps composition and separation
- PGAS useful abstraction
  - Global barrier to disappear. Locally synchronous computation encapsulated in modules and phases
- Collective behaviours more interesting than collective communications
Current work

- Semantic of patterns by way of task-based formalism
  - Capturing Data Analytics patterns
  - Spark, Flink, Storm, Tensorflow, …
  - Tasks used as a way to describe a parallel semantics for patterns

- Collective behaviours more interesting than collective communications
  - Behaviours (algorithms) need data structures
  - Embed locality awareness, redundancy schema and consistency into distributed data structures
    - Somehow already happened in C++11
    - Sort of RDD with rich metadata
bool push(void const *data) {
    unsigned long pw, seq;
    element_t *node;
    unsigned long bk = BACKOFF_MIN;
    do {
        pw = pwrite.load(std::memory_order_relaxed);
        node = &buf[pw & mask];
        seq = node->seq.load(std::memory_order_acquire);
        if (pw == seq) {
            if (pwrite.compare_exchange_weak(pw, pw+1, std::memory_order_relaxed))
                break;
            for (volatile unsigned i=0; i<bk; ++i);
            bk <<= 1;
            bk &= BACKOFF_MAX;
        } else
            if (pw > seq) return false; // queue full
    } while(1);
    node->data = data;
    node->seq.store(seq+1, std::memory_order_release);
    return true;
}
Acknowledgements

- REPARA (EC-STREP, 7th FP): Reengineering and Enabling Performance And poweR of Applications (2013, 36 months, total cost 3.5M €).
- cHiPSet (EC-COST Action IC1406): High-Performance Modelling and Simulation for Big Data Applications (2015, 48 months, total cost 500K €).
- IBM Faculty award & Joint Study Agreement: Apache Spark optimisation (est. 2015)
- Noesis Solutions: Parallel machine learning techniques for engineering (est. 2015)
- NVidia Corp: CUDA Research Center at University of Torino (est. 2013)