Supporting general data structures and execution models in runtime environments

PhD. Dissertation

Javier Fresno Bausela

Advisor: Dr. Arturo González Escribano

September 21st, 2015
Outline

Introduction

The Hitmap library

Unified support for dense and sparse data

A portable dataflow model and framework

Conclusions
Introduction
Parallel computing

● **What?**
  The simultaneous use of multiple computational resources to solve a problem.

● **Why?**
  Many computing problems are so costly that they cannot be solved sequentially in a reasonable time.

● **Where?**
  Usually associated to high-performance computing but nowadays also for mainstream computing.
The evolution of parallel computing systems

1997
ASCi Red breaks teraflop barrier.

1993
Top500 project creation.

1990's
2000's
2010's

1 teraflops
10 teraflops
100 teraflops
1 petaflops
10 petaflops
100 gigaflops

2008
IBM's Roadrunner reaches petaflops mark.

2013
Tianhe-2 - 33.86 petaflops.

Exascale

Supporting general data structures and execution models in runtime environments
Common tools for parallel computing

Google Scholar results for "parallel computing" + term

Year


Number of results (normalized 2014)

MPI, PVM, OpenMP, Pthreads, CUDA, OpenCL, PGAS, Skeletons

Supporting general data structures and execution models in runtime environments
Common tools for parallel computing II

Google Scholar search:

\[
\frac{\text{TERM} + \text{“parallel computing”}}{\text{“parallel computing”}} \%
\]

Most popular parallel tools in 2014:
- MPI 18%
- CUDA 17%
- OpenMP 12%

The most cited parallel programming tools are message-passing for distributed-memory, threads models for shared-memory environments, or kernel solutions for accelerators.
How do we currently develop programs for these systems?

We need to know:

- Sequential programming
How do we currently develop programs for these systems?

We need to know:
- Sequential programming
- Distributed memory
How do we currently develop programs for these systems?

We need to know:
- Sequential programming
- Distributed memory
- Shared memory
How do we currently develop programs for these systems?

We need to know:

- Sequential programming
- Distributed memory
- Shared memory
- Accelerator offloading

Supporting general data structures and execution models in runtime environments
How do we currently develop programs for these systems?

We need to know:

- Sequential programming
- Distributed memory
- Shared memory
- Accelerator offloading

A programmer must be proficient in all these technologies to be able to take advantage of the current parallel systems.
Development of parallel programs: How should it be?

- We need:
  - Frameworks with unified parallel models.
  - High-level abstractions to represent parallel algorithms.

- So:
  - Programmers can focus on the design
  - while compilers do the complex optimizations
  - using highly-efficient and adaptable runtime systems.
Related work

- Compiler auto-parallelization:
  - High Performance Fortran \(^1\).
  - Polyhedral model, e.g. Pluto \(^2\).

- Multi-paradigm models:
  - Partitioned Global Address Space languages: e.g. Chapel \(^3\).
  - Heterogeneous platforms: e.g. OpenCL \(^4\).

---

\(^2\) PLUTO+: near-complete modeling of affine transformations for parallelism and locality, Acharya and Bondhugula, ACM PPoPP, 2015.
\(^3\) User-defined distributions and layouts in chapel, Chamberlain et al, HotPar 2010.
\(^4\) The OpenCL specification, Khronos group, 2008.
Work carried out by Trasgo group

- **Trasgo programming framework:** A modular parallel programming framework. Its model is based on high-level, nested-parallel specifications. The high-level parallel code is transformed into a source code with Hitmap calls.

- **Hitmap runtime library:**
  - A library for hierarchical tiling and mapping of arrays.
  - Provides a global view of the parallel computation.
  - Module system to perform data partition.
  - Communications are adapted based on the partition.

---

5Trasgo: a nested-parallel programming system, Gonzalez-Escribano et al, Springer JoS, 2009 (see Ref. [58])

6An Extensible System for Multilevel Automatic Data Partition and Mapping, Gonzalez-Escribano et al, IEEE TPDS, 2013 (see Ref. [59])
Trasgo framework architecture

Program representations
- High level source code
- Intermediate representation
- Mapped program
- Target code + Hitmap calls
- Binary executable

Transformations
- Font-end translator
- Expression builder
- Back-end
- Native compiler

Supporting general data structures and execution models in runtime environments
Towards a unified programming model

Most parallel program models, including Hitmap, suffer from some limitations.

- Unified support for dense and sparse data.
- Integration of dynamic parallel paradigms and models.
Limitations I: Sparse support

- Common parallel tools do not offer integrate support for data structures.
  - MPI and OpenMP only give parallelism support.
- Most parallel languages offer support only for dense structures.
  - Such as HPF, UPC
- Some PGAS languages are being augmented with sparse support:
  - E.g. Chapel, Titanium.
- For sparse structures:
  - Manual management: implied a high programming effort.
  - Specific libraries: may not follow the same approach.
- Reusability of dense code was rather poor.
Limitations II: Dataflow structures

With common parallel solutions (e.g. MPI, OpenMP):

- Simple static parallel structures are easy to program.
- Programming dynamic and dataflow applications is still challenging.
- Low abstraction level to deal with complex synchronization:
  - Complex codes with many hard-wired decisions.
Research question

Is it possible to create a runtime system for a generic high level programming language that offers (1) common abstractions for dense and sparse data management, and (2) generic data-mapping and data-flow parallelism support for hybrid shared- and distributed-memory environments?
The Hitmap library

Supporting general data structures and execution models in runtime environments
The Original Hitmap library

- Library for hierarchical tiling and mapping of arrays.

- Main features:
  - Use of a global view of the parallel computation.
  - Module systems of load-balancing and distribution techniques.
  - Communications are declared based on partition result.
Features and terminology

- Three categories and six entities:
  - Tiling arrays
  - Mapping
  - Communication
  - Shape
  - Topology
  - Layout
  - Tile
  - Communication
  - Pattern

Supporting general data structures and execution models in runtime environments
Tiling example

A \[0:4][1:7\]

B = A[1:3][1:3]

C = B[0:1][1:2]

D = A[0:4:2][6:8:2]
Mapping example

Virtual topology: 1D

Virtual topology: 2D

Supporting general data structures and execution models in runtime environments
Communication

- Transmission of tile elements among virtual processors.
- Types: point-to-point communications, paired exchanges for neighbors, shifts along a virtual topology axis, collective communications, etc.
- Use of layouts information about neighborhood.
- Composed in reusable patterns.
The original Hitmap library architecture

Supporting general data structures and execution models in runtime environments
Supporting general data structures and execution models in runtime environments
Unified support for dense and sparse data
Sparse support in parallel frameworks

- Common parallel tools do not offer integrate support for data structures.
  - MPI and OpenMP only give parallelism support.
- Most parallel frameworks only integrate support for dense structures:
  - The Partitioned Global Address Space languages: UPC\(^7\), Coarray Fortran\(^8\).
- Some frameworks have a limited sparse support:
  - Titanium\(^9\): Sparse Array Copying.
  - Chapel\(^{10}\): Sparse domain distribution.

---

7 Introduction to UPC, Carlson et al, Tech. rep. CCS-TR-99-157, 1999 (see Ref. [23])
8 Fortran 2008 standard, ISO/IEC 2010 (see Ref. [78])
10 User-defined distributions and layouts in Chapel, Chamberlain et al, HotPar 2010 (Ref. [27])
Alternatives

- Manual management: implies a high programming effort.
- Specific libraries: may not follow the same approach.
  - Sparse management libraries: Sparskit 11.
  - Sparse partitioning tools: Metis 12.
  - Mathematical solver libraries: PETSc 13.

---

11 SPARSKIT: a basic tool kit for sparse matrix computations, Saad, Tech. rep. 1994, (Ref. [109])
12 MeTiS–A Software for Partitioning Graphs, Karypis et al, Tech. rep. 1998, (see Ref. [80])
Our proposal

- We present a solution to handle sparse and dense data domains using the same conceptual approach.

- Stages of a parallel program:
  - Sparse/Dense parallel design follows the same steps.
  - The differences appear at the implementation stage.
Stages of a parallel program: Stencil example

Supporting general data structures and execution models in runtime environments
Stages of a parallel program: Stencil example

- **Domain definition**
  - Dense
  - Sparse

- **Domain partition**

- **Memory allocation**

- **Communication**

Supporting general data structures and execution models in runtime environments
Stages of a parallel program: Stencil example

Dense
Domain definition

Sparse
Domain partition

Graph
Memory allocation

Communication

Supporting general data structures and execution models in runtime environments
Adding support for sparse domains to Hitmap

- New step in the Hitmap programming methodology.

Supporting general data structures and execution models in runtime environments
Adding support for sparse domains to Hitmap

- New step in the Hitmap programming methodology.

**Problem**
- Abstract solution
- Parallel algorithm design
- Implementation
- Select data format
- Partition technique
- Compile and run
Adding support for sparse domains to Hitmap

- New step in the Hitmap programming methodology.
- Shape and Tile classes in abstract interfaces.
- Two new kinds of sparse domains: CSR, Bitmap
- Tiles with several data spaces: edges and vertices.
- New layouts with graph partitioning.
- New communications.

Supporting general data structures and execution models in runtime environments
New architecture

Supporting general data structures and execution models in runtime environments
Programing with Hitmap dense/sparse support

Abstract Design
- Virtual Topology
- Data domain
- Local computation
- Comm. structure

Implementation
- Particular data format
- Partition technique

Executable
- Adapts at run-time depending on the real topology

Supporting general data structures and execution models in runtime environments
Dense example: distributed matrix initialization

// Load the global matrix.
HitShape sglobal = hit_shapeStd(2,ROWS,COLS);

// Create the topology object.
HitTopology topo = hit_topology(plug_topArray2D);

// Distribute the matrix among the processors.
HitLayout lay = hit_layout(layBlocks,topo,&sglobal);

// Get the shape for the local matrix.
HitShape shape = hit_layShape(lay);

// Allocate the matrix.
HitTile_double M;
hit_tileDomainShapeAlloc(&M, double, shape);

// Init the matrix values.
int i,j;
hit_shapeIterator(j,shape,0){
    hit_shapeIterator(j,shape,1){
        hit_tileElemAt(2,M,i,j) = 0.0;
    }
}

Supporting general data structures and execution models in runtime environments
Sparse example: distributed graph initialization

// Load the global matrix.
HitShape sglobal = hit_fileHBMatrixRead("file.rb");

// Create the topology object.
HitTopology topo = hit_topology(plug_topPlain);

// Distribute the matrix among the processors.
HitLayout lay = hit_layout(laySparse, topo, &sglobal);

// Get the shape for the local matrix.
HitShape shape = hit_layShape(lay);

// Allocate the matrix.
HitTile_double M;
hit_mcTileDomainShapeAlloc(&M, double, shape);

// Init the matrix values.
int i, j;
hit_cShapeRowIterator(i, shape){
    hit_cShapeColumnIterator(j, shape, i){
        hit_mcTileElemIteratorAt(M, i, j) = 0.0;
    }
}

Supporting general data structures and execution models in runtime environments
Experimental evaluation

- Three benchmarks:
  - Graph synchronization: Stencil-type operation in a graph.
  - Sparse matrix-vector multiplication.
  - Finite Element Method.

- Implementations:
  - Manual C+MPI.
  - Hitmap.
  - PETSc.

- Computing environments:
  - Geopar: A shared-memory system with 16 cores.
  - Beowulf DC: A cluster with 20 dual-core nodes.
  - Beowulf SC: A cluster with 19 single-core nodes.

Only most relevant result follow.
Results graph synchronization

GS Bodyy6 (Shared-memory system)

GS Pwt (Shared-memory system)

Execution time

Processors

Supporting general data structures and execution models in runtime environments
Results matrix multiplication

MV human_gene2 (Dual-core cluster)

MV human_gene2 (Shared-memory)

Supporting general data structures and execution models in runtime environments
Results Finite Element Method

FEM lung2 (Dual-core cluster)

FEM lung2 (Shared-memory system)

Execution time

Processors

Hitmap
PETSc
Manual C

oversubscription

Supporting general data structures and execution models in runtime environments
Lines of code comparison

MV benchmark

FEM benchmark

GS benchmark

Supporting general data structures and execution models in runtime environments
Support for sparse domains: Conclusions

- A new approach to integrate dense and sparse data management in parallel programming.
- The communication structure adapts to the data structure and partition technique.
- Hitmap abstractions simplify the writing of a parallel program with a similar performance compared to other solutions.
- The runtime for our generic parallel system now supports dense and sparse programs with the same methodology.
A portable dataflow model and framework
Stream and dataflow libraries and languages

- Programming dynamic and dataflow applications is challenging with current parallel solutions.
- Stream and dataflow: FastFlow\textsuperscript{14}, OpenStream\textsuperscript{15}, or S-Net\textsuperscript{16}.
- They have models where sequential computation and the synchronization are defined separately.
- These models lack a generic system to represent:
  - Channels with generic loops.
  - Mechanisms to express task-to-task affinities.
- There are some applications that cannot be built.

\textsuperscript{14}FastFlow: high-level and efficient streaming on multi-core, Aldinucci et al. (see Ref. [5])
\textsuperscript{15}OpenStream: Expressiveness and Data-Flow Compilation, Pop et al., (see Ref. [103])
\textsuperscript{16}A Gentle Introduction to S-Net, Grelck et al., Parallel Process. Lett. 2008, (see Ref. [64])
A portable dataflow model and framework

- We propose a new parallel programming model based on dataflow computations.
- Can be modelled using Colored Petri nets\(^\text{17}\).
- Hitmap++: A supplement to the static communication structures available in Hitmap.

\[^{17}\text{Coloured Petri nets: modeling and validation of concurrent systems, Jensen and Kristensen, Springer 2009.}\]
Our proposal

- Program: reconfigurable network of activities and typed data containers.
- MPMC channels with a work-stealing mechanism.
- Task-to-task affinity to exploit data locality.
- Single representation for shared and distributed memory.
Petri nets

A mathematical modeling language to describe systems.

Directed bipartite graph:
- Places and Transitions connected by Arcs.
- Places are marked with Tokens.
- A transition removes tokens from its input places and adds tokens to its output places.

Colored Petri nets is an extension that adds data type primitives and the ability of writing transitions with different behaviors (for each type).
Mode-driven model formulation

- The modes are the transition states and they define a configuration of I/O channels.
- Used to:
  - Define mutually exclusive tasks inside a transition.
  - Exploit data locality.
  - Reconfigure the network.
- Transitions read tokens with the color of their current mode.
- Signal system:
  - Mode-change signal: Special token to mark a mode change.
  - A mode-change propagates the signals across the network.
Mode example

Network creation

Transition A  Transition B  Transition C

Transition D
Supporting general data structures and execution models in runtime environments
Mode example

Network execution

Transition A  Transition B  Transition C  Transition D

Supporting general data structures and execution models in runtime environments
Mode example

Network execution

Transition A  Transition B  Transition C  Transition D

Supporting general data structures and execution models in runtime environments
Mode example

Network execution

Transition A  Transition B  Transition C

Transition D
Mode example

Network execution

Transition A  Transition B  Transition C  Transition D

Supporting general data structures and execution models in runtime environments
Modes to define data locality

Two-phased wavefront computation:

Network without modes

Network with modes

Supporting general data structures and execution models in runtime environments
Programming with Hitmap++

- Framework prototype: MPI + Pthreads.

- How to use it?
  - Build the transitions extending the base class.
  - Create the network connecting transitions and places.
Building transitions - Example

class MyTransition2: public Transition {
public:
    void execute()
    {
        double d1, d2; int i1, i2;
        get(&d1, &i1); // Get one pair of tokens
        get(&d2, &i2); // Get other pair

        double result = process(d1,d2,i1,i2);
        // Send a token to a particular place
        if(result > 0)
            put(&result,"place1");
        else
            put(&result,"place2");
    }
};
Building the network - Example

Place<\texttt{double}> \texttt{placeA}, \texttt{placeB}; // Declare the places
\texttt{placeA.setMaxSize}(10); // Set the place size

\texttt{MyTransition} \texttt{transition};

// Add the method and places to the default mode
\texttt{transition.addMethod}(&\texttt{MyTransition::execute});
\texttt{transition.addInput}(&\texttt{placeA});
\texttt{transition.addOutput}(&\texttt{placeB});
...

\texttt{Net net}; // Declare the net
\texttt{net.add}(&\texttt{transition}); // Add the transition
\texttt{net.run}(); // Run the net
Experimental evaluation

- **Benchmarks:**
  - Smith Waterman (Swps3)
  - Cellular Automata

- **Implementations:**
  - Reference (shared-memory) (see Ref. [122]) / Manual C+MPI
  - Hitmap
  - FastFlow, FastFlow distributed extension

- **Computing environments:**
  - Atlas: A shared-memory system with 64 cores.
  - CETA-Ciemat: A cluster with quad-core nodes.

Only most relevant result follow.
Supporting general data structures and execution models in runtime environments
Cellular Automata

Supporting general data structures and execution models in runtime environments
Lines of code comparison

C
FastFlow
distributed FastFlow
Hitmap++

swps3

Supporting general data structures and execution models in runtime environments
A portable dataflow model: Conclusions

- A new parallel programming model and framework based on the dataflow paradigm.
- Solves limitation of other proposals:
  - General MPMC system, with loops, and reconfigurable networks.
  - Transparently targets hybrid shared- and distributed-memory platforms.
- This framework extends the Hitmap library.
Conclusions
This PhD. Thesis answers the research question affirmatively.

It is possible to create a runtime system for a generic high level programming language that offers (1) common abstractions for dense and sparse data management, and (2) generic data-mapping and data-flow parallelism support for hybrid shared- and distributed-memory environments.
Thesis conclusions

- This Ph.D. Thesis gives an answer to these problems:
  - The unified support for dense and sparse data.
  - The integration of data-mapping and data-flow parallelism.
- Our implementation extends the Hitmap library:
  - To support dense and sparse data structures.
  - With a model for dataflow mechanisms.
Contributions I

Our first step: Study of Hitmap automatic data-layout techniques applied to multigrid methods.

- Journal article:

- Conference article:
Contributions II
Integration of dense and sparse data support into Hitmap.

- Journal articles:

- Conference and workshop articles:
Contributions III

A new model for dataflow mechanisms.

- Conference article:
  - Fresno, Gonzalez-Escribano and Llanos. “Runtime Support for Dynamic Skeletons Implementation”. *Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA).* 2013.

- Research stay:
  - —. “Exploiting parallel skeletons in an all-purpose parallel programming system”. *Science and Supercomputing in Europe - research highlights (HPC-Europa2 project).* 2012.
  - —. “Dataflow Programming Model for Hybrid Distributed and Shared Memory Systems”. Work in progress for a journal publication.
Future directions

- **Higher-level abstraction artifacts:**
  - Specialized networks.
  - Skeletons.

- **Development of new mapping policies:**
  - Load balancing.
  - Heterogeneous systems.

- **Transformation from high-level code:**
  - Open issue: Data structure, topology, and layout selection.
Thanks
Supporting general data structures and execution models in runtime environments

PhD. Dissertation

Javier Fresno Bausela

Advisor: Dr. Arturo González Escribano

September 21st, 2015