Research Experience
Energy-Efficient Computing Vision
Case Study: Many-core

Pedro Valero-Lara

CFD & Computational Technology, Basque Center for Applied Mathematics (BCAM)

September 18, 2015
Motivation

- Research Experience (Main Goals)
- Energy-Efficient Computing Vision
- Current and Upcoming Projects
Research Experience (Main Goals)
Alztool 3D Slicer Project
Motivation
  ▶ Developing software that adapted the space routines to analyse human brain scans and prevent Alzheimer diseases.

Partners:
  ▶ Deimos-Space, UNED, UCLM, Technical University of Madrid, ...

My contribution:
  ▶ Accelerating image processing by using GPUs
  ▶ ITK-MRF, OTB-MRF, SPM-DARTEL
Alztool 3D Slicer Project - MRF
Alztool 3D Slicer Project - MRF
Markov Random Field (MRF) Filter

- ITK: Insight Toolkit (Brain MRI images)

- Orfeo ToolBox (Satellite images)
Algorithm 2 Pseudocode of GPU-CMRF algorithm

Function $I_{\text{Class}} = \text{MSFImageFilter}(I_{\text{MRI}}, I_{\text{Label}}, \text{Classes}, \text{ConVar}, \text{Window})$

Inputs:
- $I_{\text{MRI}}$: MRI image
- $I_{\text{Label}}$: Labeled image
- Classes: Classes considered to classify
- ConVar: Number of pixels changed at the previous iteration
- Window: Size of the window

Output:
- $I_{\text{Class}}$: Classified image

1. while (do(iter < MAX_ITER) and (Error > MAX_ERROR))
2. Compute the influence of each Class on the central pixel of the Window over $I_{\text{MRI}}$.
3. Compute the influence of each Class of all the pixels of the Window around the central pixel of the $I_{\text{MRI}}$.
4. Compute the Mahalanobis Distance of the central pixel of the Window of the $I_{\text{MRI}}$.
5. Compute the new label of the image, according to $I_{\text{Label}}$ and the Mahalanobis Distance previously computed.
6. Compute Error
7. end while
Alztool 3D Slicer Project - Performance MRF

**ITK**
- **Speedup**
  - Graph showing speedup vs. windows size with lines for different window sizes (3x3, 5x5, 7x7).

**OTB**
- **Speedup**
  - Graph showing speedup vs. number of classes with lines for different number of classes (3, 4, 5).

**Power Consumption**
- Graph showing power consumption vs. time for different processors (GTX285, Core2Quad, 9600GT, Core2Duo) with lines for sequential and parallel execution.

- Average sequential and parallel execution times are also shown.
Alztool 3D Slicer Project - DARTEL
Diffeomorphic Anatomical **Registration** Through Exponentiated Lie algebra algorithm

![SPM](image)

- Statistical Parametric Mapping (SPM)
- Template Generation
- Warped Image Generation
Alztool 3D Slicer Project - GPU DARTEL

GPU Processing:

- Small Deformation (5%)
- Jacobian Composition (40%)
- Full Multigrid Grid (30%)

Small Deformations

Threads Id. 0 1 2
Memory Locations 0 1 2
Pixels

Jacobian Composition

0 1 2
0
0 1 2
3 4 5
6 7 8

Deformation Matrices

FMG (odd case)

0 1 2
0
0 1 2
3 4 5
6 7 8

Deformation Matrices

loop i = 0:N−1
loop b=⌊iB+(i+1)B⌋−1
loop c=⌊iB+(i+1)B⌋−1
loop j=0:M−1

loop i = 0:N−1
loop j=0:M−1
loop k=0:L−1
loop l=0:H−1
loop m=0:K−1
loop n=0:T−1
loop p=0:O−1
loop q=0:S−1
loop r=0:N−1
Alztool 3D Slicer Project - Performance DARTEL

Template Generation

Time

Power Consumption

Warped Generation

Time

Power Consumption

GPU Speedup:

- Small Deformation (32×)
- Jacobian Composition (19×)
- Full Multigrid Grid (9×)
Similarity Search
Similarity Search

Motivation

- Developing new parallel similarity search methods over GPUs

Partners:

- University of Magallanes (Chile), University of Castilla-La Mancha (Spain), Yahoo

My contribution:

- Metric Spaces - Spaghettis Data Structure
- Longest Common Subsequences (LCS)
Similarity Search - Spaghetttis Data Structure
Similarity Search - Spaghettis Data Structure

![Diagram of Spaghettis Data Structure]

**Search Total Cost, Sequential vs Parallel (n=77,455 words)**

- Seq. Brute Force
- Seq. 04
- Seq. 08
- Seq. 16
- Seq. 32

**Brute Force (gpu)**
- P 04 (gpu)
- P 08 (gpu)
- P 16 (gpu)
- P 32 (gpu)

**Speed-up for Search (n=77,455 words; Spanish dictionary)**

- Brute Force (gpu)
- Pivots : 04
- Pivots : 08
- Pivots : 16
- Pivots : 32
Similarity Search - Longest Common Subsequence (LCS)
Similarity Search - Longest Common Subsequence (LCS)

The most used similarity search method
High data dependence

Applications

Large sequences (Millions)
  Biological-DNA sequences
Small sequences (Dozens)
  Metric Spaces (words)
Medium sequences (Hundreds/Thousands)
  Data compression
  File comparison
  Intrusion detection, . . .

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>E</th>
<th>I</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

ALCS

RLCS
Similarity Search - GPU LCS

For $i=1$ to $(M-1)$

First Phase

Second Phase
Similarity Search - Performance LCS

### # sequences

![Graph showing the performance of LCS with respect to the number of sequences.](image1)

---

### Left sequences

![Graph showing the performance of LCS with respect to the left sequence size.](image2)

---

### Top sequences

![Graph showing the performance of LCS with respect to the top sequence size.](image3)
Multi Tasking (MT) on GPUs
Multi Tasking (MT) on GPUs

Motivation

- Analyze the performance of current GPUs for MT
- Developing new approaches more efficient

Approaches

- Merge
- Concurrent
- Dynamic

Full-Overlapped Concurrent Kernels
Performance MT

**Tesla vs Fermi**

![Graph comparing GTX 480 Merge, GTX 480 Concurrent, and GTX 285 Merge performance vs matrix size.](image)

**Kepler Performance**

![Graph showing Kepler performance vs matrices size.](image)

**Full-Overlapped Concurrent Kernels on Kepler**

![Diagram of full-overlapped concurrent kernels on Kepler.](image)
Supercomputing and eScience (SyeC) Project
Supercomputing and eScience (SyeC) Project

Motivation

- Advancement of supercomputing through collaboration between users and designers of hardware and software for supercomputers.

Partners:

- BSC, University of Zaragoza, University Complutense of Madrid, CIEMAT, Barcelona University, ...

My contribution:

- Developing and Acceleration of Computational Fluid Dynamics (CFD) Solvers
- Fast Poisson Problems (BLKTRI), Lattice-Boltzmann Method, Immersed-Boundary
Supercomputing and eScience (SyeC) Project - Fast Poisson Problem (BLKTRI)
SyeC Project - Fast Poisson Problem (BLKTRI)

- Accelerating fast Poisson solver on heterogeneous architectures
  - Implemented in the FORTRAN open-source package FISHPACK (BLKTRI)
  - Main bottleneck in many fluid simulation
- This method resolves finite difference discretization of 2D separable elliptic equations:
  \[ \frac{\partial}{\partial u} \left( a(u) \frac{\partial x}{\partial u} \right) + b(u) \frac{\partial x}{\partial u} + c(u)u + \frac{\partial}{\partial v} \left( d(v) \frac{\partial x}{\partial v} \right) + e(v) \frac{\partial x}{\partial v} + f(v)u = g(u, v) \]
- Using Dirichlet or Neuman boundary conditions we obtain a linear system \( \mathbf{A} \tilde{x} = \tilde{g} \):

\[
\mathbf{A} = \begin{bmatrix}
B_1 & C_1 & & & \\
A_2 & B_2 & C_2 & & \\
& \ddots & \ddots & \ddots & \\
& & A_{n-1} & B_{n-1} & C_{n-1} \\
& & & A_n & B_n
\end{bmatrix}
\]

\[ A_i = a_i I, B_i = B + b_i I, C_i = c_i I \]

where \( I \) is a identity matrix, \( a_i, b_i, c_i \) are scalars and \( B \) is a tridiagonal matrix
SyeC Project - Parallel BLKTRI

Dynamic Parallelism

- Reduction
- Substitution

Tridiagonal System

![Diagram of a tridiagonal system with numbers and arrows connecting them.](image-url)
SyeC Project - Performance BLKTRI

Heterogeneous Approach

- High → GPU
- Low → Multicore

Performance
Supercomputing and eScience (SyeC) Project - Lattice Boltzmann Method (LBM) & Immersed Boundary (IB)
SyeC Project - Lattice Boltzmann Method (LBM) & Immersed Boundary (IB)

Developing and accelerating Fluid-Solid Interaction (FSI) solvers on heterogeneous architectures

▶ LBM (Fluid) - IB (Solid) Coupling

\[
\frac{\partial f}{\partial t} + e \nabla f = \Omega + F
\]
SyeC Project - Parallel LBM & IB

Homogeneous GPU Approach

Homogeneous GPU

Heterogeneous Multicore-GPU Approach

Heterogeneous GPU–Multicore
Homogeneous

Heterogeneous

MFLUPS

Number of nodes × 10⁶

- LBM
- LBM–IB (0.5%)
- LBM–IB (1%)

MFLUPS

Number of nodes × 10⁶

- GPU
- MULTICORE
LBM-HPC Framework
Motivation
Developing an open-source and multiplatform LBM framework for HPC

- LBM-HPC

- http://www.bcamath.org/es/research/lines/CFDCT/software
LBM-HPC Framework - PGAS

- Shared memory view of distributed memory systems
- Facilitate parallel implementations
- Provide better efficiency and scalability
- Hide important aspects regarding parallel programming
  - One-side communication
  - No communication calls
  - No buffering
  - Transparent workload distribution, ... 
- Multiple languages
  - Coarray Fortran
  - Unified Parallel C (UPC)
  - Chapel
- API
  - The Global address space Programming Interface (GPI)
  - Global Arrays Toolkit
LBM-HPC Framework - UPC

- Extension of C
- Two thread monitoring:
  - THREADS, number of UPC threads
  - MYTHREAD, gives the index of the current thread
- Declare array accessible by all UPC threads
  - shared
- Affinity → block-size qualifier
  - shared[ ] type array[size] Implicit
  - shared[block-size] type array[size] Explicit
  - upc forall(initial;test;increment;affinity) Implicit
- affinity:
  - pointer
  - integer expression (affinity%THREADS)
LBM-HPC Framework - Approaches

Hybrid MPI-OpenMP
- OpenMP pragmas for LBM solver
- MPI_send/receive for communication
- Ghost cells

UPC Implicit
- LBM and communication packed
- Transparent
- No ghost cells

UPC Explicit
- Explicit communication
- Ghost cells
  - No buffers
  - No specific calls
// #DEFINE CHUNK Lx*Ly/THREADS
int main(int argc, char **argv){
    static shared[CHUNK] double u[Ly][Lx],v,p;
    static shared[CHUNK] double f1[Ly][Lx][9],f2;
    t=0;
    while(t<It)
    {
        /*——— Boundary Conditions ———*/
        upc forall(i=0; i<Ly; i++; &u[i][0])
        {
            for(j=0; j<Lx; j++)
            {
            // Top boundary
                if(i==0)
                {
                    u[i][j] = uMax;
                    v[i][j] = v0;
                    p[i][j] = p0;
                    for(z=0; z<9; z++)
                    {
                        cu=3*(cx[z]*uMax+cy[z]*v0);
                        -(3./2.*((uMax)^2+(v0)^2));
                        f1[i][j][z] = f;
                    }
                }
                // Other boundary conditions
            }
        }
        /*——— LBM and Communication ———*/
        upc forall(i=1; i<Ly-1; i++; &u[i][0])
        {
            for(j=1; j<Lx-1; j++)
            {
                u_local=0.;v_local=0.;p_local=0.;
                for(z=0; z<9; z++)
                {
                    new_i=i-cx[z]; new_j=j-cy[z];
                    ftmp[z]= f1[new_i][new_j][z];
                }
            }
            t++;
        }
    }
}
LBM-HPC - Explicit UPC

```c
// UPC_x(y) UPC threads in x(y)-direction
// TH = THREADS
// MT = MYTHREAD
int main(int argc, char **argv) {
    mid_x = MT%UPC_x;
    mid_y = floor(MT/UPC_x);
    static shared [*] double u[TH][Ly][Lx], v, p;
    static shared [*] double f1[TH][Ly][Lx][9], f2;
    t=0;
    while(t<It) {
        /*——— Boundary Conditions ———*/
        // Top boundary
        if(mid_y == 0) {
            for(j=0; j<Lx; j++) {
                u[MT][0][j] = uMax;
                v[MT][0][j] = v0;
                p[MT][0][j] = p0;
                for(z=0; z<9; z++) {
                    cu=3*(cx[z]*uMax+cy[z]*v0);
                    f=p0*w[z]*(1.+cu+1./2.*(cu)^2)
                        -(3./2.*((uMax)^2+(v0)^2));
                    f1[MT][0][j][z] = f;
                }
            }
        }
        /*——— Communication ———*/
        // From bottom to top
        if(mid_y > 0 && UPC_y > 1) {
            for(j=0; j<Lx; j++) {
                for(z=0; z<9; z++) {
                    f1[MT][0][j][z]=f1[MT-UPC_x][Ly-2][j][z];
                }
            }
        }
        /*——— LBM ———*/
        for(i=1; i<Ly-1; i++) {
            for(j=1; j<Lx-1; j++) {
                u_local=0.; v_local=0.; p_local=0.;
                for(z=0; z<9; z++) {
                    new_i=i-cx[z]; new_j=j-cy[z];
                    ftmp[z] = f1[MT][new_i][new_j][z];
                }
                t++;
            }
        }
    }
    /*——— Other boundary conditions
    ...// Other directions
```
LBM-HPC - Platform

High Performance Computing Center Stuttgart (HLRS) of the University of Stuttgart

<table>
<thead>
<tr>
<th>Platform Cabinets</th>
<th>Hornet (Cray XC40)</th>
</tr>
</thead>
<tbody>
<tr>
<td># Compute nodes</td>
<td>21</td>
</tr>
<tr>
<td># Compute cores</td>
<td>3944</td>
</tr>
<tr>
<td># Processor</td>
<td>94656 (24 cores per node)</td>
</tr>
<tr>
<td></td>
<td>Intel Xeon CPU E5-2680 v3</td>
</tr>
<tr>
<td></td>
<td>(30M Cache, 2.50 GHz)</td>
</tr>
<tr>
<td></td>
<td>12</td>
</tr>
<tr>
<td># cores/processor</td>
<td>5.4 PB (128 GB per node)</td>
</tr>
<tr>
<td>Total compute memory</td>
<td>Cray Aries (Dragonfly topology)</td>
</tr>
<tr>
<td>Node-node interconnect</td>
<td>3786 TeraFLOPS</td>
</tr>
<tr>
<td>Peak performance (23\textsuperscript{th} in TOP 500)</td>
<td></td>
</tr>
</tbody>
</table>
LBM-HPC - Case Study

- Lid-driven cavity
- Performance Analysis
  - Boundary Conditions
  - Communication
  - LBM solver
- Size problem $\equiv 1000$ million of fluid nodes
- Double precision
- Strong scaling and time
- 240 - 30720 cores
LBM-HPC Performance

- **MPI-OpenMP**
  - The biggest communication overhead (2% → 7.5%)

- **UPC Explicit**
  - Low communication overhead (0.3% → 1.8%)
  - Equivalent BCs overhead with respect to MPI-OpenMP

- **UPC Implicit**
  - Largest BCs overhead
  - Main bottleneck over high number of cores
Energy-Efficient Computing Vision
Case Study: Many-core
Energy-Efficient Computing Vision
Case Study: Many-core

Big cores
- ILP does not scale either
- Complex Circuits
  - Branch Prediction
  - Out of Order Execution
  - Re-order Buffer
- Big Memory Caches
- Low-latency oriented
- High Watt/Performance

Small cores
- No complex circuits
- Small (or no) cache
- Throughput oriented
- Low Watt/Performance
Energy-Efficient Computing Vision

Case Study: Many-core

GPU (K80)
- 4992 CUDA cores
- Concurrency
- Shared Memory
- Small Caches
- Coalescing

MIC (7120P)
- 61 512-bit vector cores
- Ring interconnect
- Big Caches
- Vectorization
Energy-Efficient Computing Vision
Case Study: Many-core

Memory transfer among memories

- Peak power consumption

Global memory

- Multiple accesses
- Concurrency (GPU)
Energy-Efficient Computing Vision
Case Study: Many-core

- Bus communication
- No concurrency
- ARM cores
- Low cost
- **Tilera**
  - 72 Cores
  - 64 bit
- **Epyphany Parallela**
  - 64 Cores
  - 1 MB cache/core
  - FPGAS
  - 32 bit
- **Kalray**
  - 256 Cores
  - 16 on-chip Clusters
  - 32 bit

Keeping data in local memory!!

Good performance for decoding video
What about HPC application?
Bigger local memory?
Current and Upcoming Projects
Current and Upcoming Projects

LBM-HPC

- Dynamic/Automatic Refinement
- Elasticity and Breaking Effect over solids
  - Immersed Boundary

LBM-HPC

- Heterogeneous Platforms
- Abel Paz, CETA-CIEMAT, Spain

LBM-HPC PGAS

- Power Consumption
- Daniel Rubio, HLRS, Germany
Current and Upcoming Projects

LBM Kalray
- Performance/Consumption Study
- Luis Pinho, CISTER, Portugal

LBM Parallela Epiphany
- Performance/Consumption Study
- Sebastian Raase, Halmstad University, Sweden

Multi-Tasking on GPU and MIC
- Performance Study and Development
- Poornima Nookala, Illinois Institute of Technology, USA
Current and Upcoming Projects

LBM-QSE (PGAS-Chapel)

- Ocean Simulation
- Erich Foster, Universit della Svizzera Italiana, Switzerland
Research Experience
Energy-Efficient Computing Vision
Case Study: Many-core

Pedro Valero-Lara

CFD & Computational Technology, Basque Center for Applied Mathematics (BCAM)

September 18, 2015