On the Autotuning of Task-Based Numerical Libraries for Heterogeneous Architectures

Jesús Cámara

Joint work with
E. Agullo, J. Cuenca and D. Giménez

International Conference on Parallel Computing
(ParCo2019)

Prague, September 2019
Overview

1. Motivation

2. Application Case

3. Search of the Block Size

4. Selection of Computing Units

5. Conclusions and Future Work
Overview

1. Motivation

2. Application Case

3. Search of the Block Size

4. Selection of Computing Units

5. Conclusions and Future Work
The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.
Motivation

The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.
The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.
The complexity of modern computational platforms makes the design of high performance numerical libraries extremely challenging.

Task-Based Libraries = Sequential Task Flow (STF) design of Linear Algebra Routines on top of Runtime Systems.
Motivation

**Chameleon**: Dense Linear Algebra Library derived from PLASMA (tile algorithms) with a STF design on top of Runtime Systems.

**Linear Algebra**

\[ AX = B \]

**Tile Matrix Layout**

```
for (j = 0; j < N; j++)
    Task(A[j]);
```

**Sequential-Task-Flow**

```
for (j = 0; j < N; j++)
    Task(A[j]);
```

**Direct Acyclic Graph**

![Direct Acyclic Graph](image)

**Runtime Systems (Scheduling of Tasks)**

![Runtime Systems](image)

**StarPU**

**Optimized Kernels**

MKL, cuBLAS, ...

**Heterogeneous Platforms**

![Heterogeneous Platforms](image)
Auto-Tuning

Process to obtain the best value for some algorithmic parameters in order to reduce the execution time of the routines with an efficient use of the heterogeneous platform.
Optimization Strategies

Auto-Tuning

Process to obtain the best value for some algorithmic parameters in order to reduce the execution time of the routines with an efficient use of the heterogeneous platform.

\[ \text{nb} = 208, 256, 288, \ldots \]

\[ n \]

\[ \text{DAG of tasks} \]
Optimization Strategies

Auto-Tuning

Process to obtain the best value for some algorithmic parameters in order to reduce the execution time of the routines with an efficient use of the heterogeneous platform.

![Diagram showing nb=208, 256, 288, ... and a task dependency graph with nodes labeled 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11.]

Performance vs. Concurrency

Trade off between the performance of a task (the higher \( nb \), the higher the performance) and the concurrency between tasks (the smaller \( nb \), the wider the DAG of tasks).
The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ... but it not takes care about the best value for some APs.

Which parameters to consider?

- Tile-Size.
- Number of Computing Units.

Other Parameters? Not addressed here!
The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ... but it not takes care about the best value for some APs.
The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ... but it not takes care about the best value for some APs.

Other Parameters?

Not adressed here!

Scheduling Policies, Threads per CPU kernel, ...
The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.

... but it not takes care about the best value for some APs.
The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ... but it not takes care about the best value for some APs.
Optimization Strategies

The RS does most of the work:
- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

Which parameters to consider?
## Optimization Strategies

**The RS does most of the work:**
- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ...but it not takes care about the best value for some APs.

**Which parameters to consider?**
- Tile-Size.
Optimization Strategies

The RS does most of the work:
- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
- ... but it not takes care about the best value for some APs.

Which parameters to consider?
- Tile-Size.
- Number of Computing Units.
The RS does most of the work:

- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.

...but it not takes care about the best value for some APs.

Which parameters to consider?

- Tile-Size.
- Number of Computing Units.

Other Parameters?

Scheduling Policies, Threads per CPU kernel, ...
Optimization Strategies

The RS does most of the work:
- Handles data consistency.
- Handles data dependencies.
- Handles scheduling of tasks to the CUs.
  ... but it not takes care about the best value for some APs.

Which parameters to consider?
- Tile-Size.
- Number of Computing Units.

Other Parameters? Not addressed here!
Scheduling Policies, Threads per CPU kernel, ...
Optimization Strategies

Goal

- Search the best $nb$ value for each problem size $n$. 

Four Strategies (S1, S2, S3, S4):
- S1: Empirical+Exhaustive.
- S2: Empirical+Pruned.
- S3: Simulated+Exhaustive.
- S4: Simulated+Pruned.
Optimization Strategies

Goal

- Search the best $nb$ value for each problem size $n$.
- Select the number of computing units to use.
Optimization Strategies

Goal

- Search the best $nb$ value for each problem size $n$.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- S1: Empirical+Exhaustive.
- S2: Empirical+Pruned.
- S3: Simulated+Exhaustive.
- S4: Simulated+Pruned.
Optimization Strategies

Goal

- Search the best \( nb \) value for each problem size \( n \).
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.
Optimization Strategies

Goal
- Search the best $nb$ value for each problem size $n$.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):
- **S1**: Empirical+Exhaustive.
- **S2**: Empirical+Pruned.
- **S3**: Simulated+Exhaustive.
- **S4**: Simulated+Pruned.
Optimization Strategies

Goal

- Search the best $nb$ value for each problem size $n$.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):

- **S1**: Empirical + Exhaustive.
- **S2**: Empirical + Pruned.
- **S3**: Simulated + Exhaustive.
- **S4**: Simulated + Pruned.
Optimization Strategies

Goal
- Search the best $nb$ value for each problem size $n$.
- Select the number of computing units to use.

Four Strategies (S1, S2, S3, S4):
- S1: Empirical+Exhaustive.
- S2: Empirical+Pruned.
- S3: Simulated+Exhaustive.
- S4: Simulated+Pruned.
Optimization Strategies

Calibration

Simulation

Valid in a wide range of settings

Scheduling details

StarPU

Performance profiles

Platform description

Many simulations at low cost!

SimGrid

<table>
<thead>
<tr>
<th>qr_mumps</th>
<th>Cores</th>
<th>RAM</th>
<th>Evaluation</th>
<th>Makespan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Native</td>
<td>40</td>
<td>58.0GiB</td>
<td>157s</td>
<td>141s</td>
</tr>
<tr>
<td>SimGrid</td>
<td>1</td>
<td>1.5GiB</td>
<td>57s</td>
<td>142s</td>
</tr>
</tbody>
</table>

Run once!
Overview

1 Motivation

2 Application Case

3 Search of the Block Size

4 Selection of Computing Units

5 Conclusions and Future Work
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW, A[i][k],
                  R, A[i][j],
                  R, A[k][j]);
        }
    }
}

GEMM
SYRK
TRSM
POTRF
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW, A[i][k],
                  R, A[i][j],
                  R, A[k][j]);
        }
    }
}
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW, A[i][k],
                  R, A[i][j],
                  R, A[k][j]);
        }
    }
}

GEMM
SYRK
TRSM
POTRF
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}

Jesús Cámara  (University of Murcia)
Tuning Task-Based Numerical Libraries  Prague, September 2019  11 / 37
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k], R,A[i][j], R,A[k][j]);
        }
    }
}
The Cholesky Algorithm

```plaintext
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW, A[i][k],
                  R, A[i][j],
                  R, A[k][j]);
        }
    }
}
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW, A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW, A[i][j], R, A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW, A[i][i], R, A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW, A[i][k],
                  R, A[i][j],
                  R, A[k][j]);
        }
    }
}

GEMM
SYRK
TRSM
POTRF
The Cholesky Algorithm

```
for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k], R,A[i][j], R,A[k][j]);
        }
    }
}
```

Jesús Cámara  (University of Murcia)  Tuning Task-Based Numerical Libraries  Prague, September 2019  11 / 37
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}

GEMM
SYRK
TRSM
POTRF

Jesús Cámara  (University of Murcia)  Tuning Task-Based Numerical Libraries  Prague, September 2019  11 / 37
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
The Cholesky Algorithm

for (j = 0; j < N; j++) {
    POTRF (RW,A[j][j]);
    for (i = j+1; i < N; i++) {
        TRSM (RW,A[i][j], R,A[j][j]);
    }
    for (i = j+1; i < N; i++) {
        SYRK (RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
The Cholesky Algorithm

Goal

Search the best $nb$ value for a given problem size $n$. 
The Cholesky Algorithm

Goal
Search the best $nb$ value for a given problem size $n$.

Which $nb$ range?
The Cholesky Algorithm

Goal
Search the best $nb$ value for a given problem size $n$.

Which $nb$ range?
- In principle any.
The Cholesky Algorithm

Goal
Search the best \textit{nb} value for a given problem size \( n \).

Which \textit{nb} range?
- In principle any.
- Only 8 values considered in this work:
  \{208, 256, 288, 320, 384, 448, 512, 576\}
Overview

1 Motivation

2 Application Case

3 Search of the Block Size

4 Selection of Computing Units

5 Conclusions and Future Work
Heterogeneous Platform

Jupiter
- 2 Intel Xeon E5-2620 (2.00 GHz) (1 x 5 cores)
- RAM: 32 GB
- L1: 32 KB, L2: 256 KB, L3: 12 MB

GPU
- GeForce GTX 590
  - 512 cores - 1.5 GB

MIC
- Xeon Phi 3120A KNC
  - 57 cores (1.2 GHz)
  - 6 GB - 28.5 MB L2

mercurio
- 4 Intel Xeon E7530 (1.87 GHz) (4 x 5 cores)
- RAM: 32 GB
- L1: 32 KB, L2: 256 KB, L3: 12 MB

GPU
- Tesla C2050 (Kepler)
  - 2496 cores - 5 GB

MIC
- Xeon Phi 3120A KNC
  - 57 cores (1.2 GHz)
  - 6 GB - 28.5 MB L2

marte
- 1 AMD Phenom II X6 1075T (800 MHz) (1 x 6 cores)
- RAM: 16 GB
- L1: 64 KB, L2: 32 KB, L3: 5 MB

GPU
- GeForce GTX 460
  - 460 cores - 1.5 GB

MIC
- Xeon Phi 3120A KNC
  - 57 cores (1.2 GHz)
  - 6 GB - 28.5 MB L2

saturno
- 4 Intel Xeon E7530 (1.87 GHz) (4 x 5 cores)
- RAM: 32 GB
- L1: 32 KB, L2: 256 KB, L3: 12 MB

GPU
- Tesla C2050 (Kepler)
  - 2496 cores - 5 GB

MIC
- Xeon Phi 3120A KNC
  - 57 cores (1.2 GHz)
  - 6 GB - 28.5 MB L2

venus
- 2 Intel Xeon E5-2620 (2.00 GHz) (1 x 5 cores)
- RAM: 64 GB
- L1: 32 KB, L2: 256 KB, L3: 15 MB

GPU
- GeForce GTX 590
  - 512 cores - 1.5 GB

MIC
- Xeon Phi 3120A KNC
  - 57 cores (1.2 GHz)
  - 6 GB - 28.5 MB L2

Tuning Task-Based Numerical Libraries
Prague, September 2019
Heterogeneous Platform

Jupiter

- 2 Intel Xeon E5-2620 Hexa-Core (12 CPU Cores).
- 4 GPU NVIDIA GeForce GTX590.
- 2 GPU NVIDIA Tesla C2075.
Performance significantly depends on the value of the Tile Size ($nb$).
Performance depends on the value of the Tile Size
S1: Empirical+Exhaustive

No. Experiments: 8 problem sizes * 8 tile sizes = 64
Which $nb$ among these 8 values for a $n$ given at runtime?
nb with a higher performance for each problem size $n$
S1: Empirical+Exhaustive

Experimental Time: 67 seconds

Matrix Order vs. GFlop/s for different block sizes nb:
- nb = 576
- nb = 512
- nb = 448
- nb = 384
- nb = 320
- nb = 288
- nb = 256
- nb = 208

(67)
Experimental Time: 138 seconds
S1: Empirical+Exhaustive

Experimental Time: 30 minutes
S2: Empirical+Pruned
S2: Empirical + Pruned

Matrix Order

GFlop/s

0 0.4 0.8 1.2 1.6 2 2.4 2.8 3.2

0 200 400 600 800 1,000 1,200

256
S2: Empirical+Pruned

Matrix Order vs. GFlop/s for S2: Empirical+Pruned. The graph shows the performance of the task-based numerical libraries with different matrix orders.

- **Matrix Order**: The x-axis represents the matrix order, ranging from 0 to 3.2 \( \times 10^4 \).
- **GFlop/s**: The y-axis represents the GFlop/s, ranging from 0 to 1,200.

The graph includes data points marked with a star and a diamond, indicating specific performance metrics for matrix orders 208.

This data was collected during the tuning of task-based numerical libraries at the University of Murcia, Prague, September 2019.
S2: Empirical+Pruned

![Graph](image-url)
S2: Empirical+Pruned

![Graph showing performance of different matrix orders]
S2: Empirical+Pruned

Matrix Order vs GFlop/s

- Matrix Order: 0, 0.4, 0.8, 1.2, 1.6, 2, 2.4, 2.8, 3.2
- GFlop/s: 0, 200, 400, 600, 800, 1000, 1200

- Points:
  - 208 GFlop/s at Matrix Order of 0
  - 256 GFlop/s at Matrix Order of 0

Jesús Cámara (University of Murcia)
S2: Empirical+Pruned

Matrix Order vs. GFlop/s for various matrix orders and their corresponding GFlop/s values.

- Matrix Order: 0, 0.4, 0.8, 1.2, 1.6, 2, 2.4, 2.8, 3.2
- GFlop/s: 0, 200, 400, 600, 800, 1000, 1200

Key Points:
- Matrix Order 208: GFlop/s 200
- Matrix Order 256: GFlop/s 400
- Matrix Order 288: GFlop/s 600

Graph illustrates performance trends across different matrix orders.
S2: Empirical+Pruned

Matrix Order

GFlop/s

0 0.4 0.8 1.2 1.6 2 2.4 2.8 3.2

·10^4

208

256

320
S2: Empirical + Pruned

Matrix Order vs. GFlop/s

- Data points for different matrix orders and GFlop/s values.

Jesús Cámara (University of Murcia)
S2: Empirical+Pruned

![Graph showing empirical and pruned performance data for matrix orders ranging from 200 to 1200 GFlop/s.]
S2: Empirical + Pruned

Matrix Order vs. GFlop/s graph:

- Matrix Orders: 208, 256, 512
- GFlop/s: 0, 200, 400, 600, 800, 1000, 1200

Jesús Cámara (University of Murcia)
Tuning Task-Based Numerical Libraries
Prague, September 2019
S2: Empirical+Pruned

Matrix Order vs. GFlop/s for different matrix orders.
S2: Empirical+Pruned

Matrix Order vs. GFlop/s from 208 to 512.

- 208 GBytes: 208 GFlop/s
- 256 GBytes: 448 GFlop/s
- 512 GBytes: 800 GFlop/s
S2: Empirical + Pruned
S2: Empirical+Pruned

Matrix Order vs. GFlop/s graph with data points at:
- 208 GFlop/s for Matrix Order 0
- 256 GFlop/s for Matrix Order 0.8
- 448 GFlop/s for Matrix Order 1.2
- 576 GFlop/s for Matrix Order 1.6

Jesús Cámara (University of Murcia)
S2: Empirical + Pruned

Experimental Time: 154 seconds
Exhaustive (S1) vs Pruned (S2)

Same \( nb \) value for each problem size \( n \)
S1: 30 min; S2: 154 seconds
S3: Simulated Exhaustive

![Graph showing GFlop/s vs. Matrix Order with nb = 208]
S3: Simulated + Exhaustive

Matrix Order

GFlop/s

$nb = 208$

$nb = 256$

- $nb = 208$

- $nb = 256$

Matrix Order $\cdot 10^4$
S3: Simulated\text{+}Exhaustive

![Graph showing GFlop/s vs Matrix Order for different block sizes nb]

- **G brothere**
  - nb = 576
  - nb = 512
  - nb = 448
  - nb = 384
  - nb = 320
  - nb = 288
  - nb = 256
  - nb = 208
Empirical (S1) vs Simulated (S3)

1a. Empirical Performance

1b. Simulated Performance
S3: Simulated + Exhaustive

Matrix Order

GFlop/s

208
256
576

nb with a higher performance for each problem size n
S3: Simulated Exhastive

Simulation Time: 2 seconds
Simulation Time: 8 seconds
S3: Simulated + Exhaustive

Simulation Time: 30 minutes
Simulation Time: 84 seconds
Exhaustive (S3) vs. Pruned (S4)

Same $nb$ value for each problem size $n$
$S3$ & $S4$

**Graph Description:**
- **Y-axis:** GFlop/s
- **X-axis:** Matrix Order
- **Legend:**
  - Green stars: Simulated + Pruned
  - Orange circles: Simulated + Exhaustive

**Data Points:**
- S3: 30 min
- S4: 84 sec

**Graph Analysis:**
- The graph shows the performance of numerical libraries across different matrix orders.
- The Simulated + Pruned method outperforms the Simulated + Exhaustive method as the matrix order increases.

---

**Credits:**
- Jesús Cámara (University of Murcia)
- Tuning Task-Based Numerical Libraries
- Prague, September 2019
S1 & S2 & S3 & S4

Matrix Order

GFlop/s

S1: 30 min; S2: 154 sec; S3: 30 min; S4: 84 sec
Overview

1 Motivation

2 Application Case

3 Search of the Block Size

4 Selection of Computing Units

5 Conclusions and Future Work
Selection of Computing Units

StarPU

Tasks are assigned dynamically to the CUs using a scheduling policy, but it does not take into account the computational power of the CUs.
Selection of Computing Units

StarPU

Tasks are assigned dynamically to the CUs using a scheduling policy, but it does not take into account the computational power of the CUs.

Goal

Decide the best number of CUs (number of CPU cores and GPUs) to use for a given problem size $n$. 
Selection of Computing Units

StarPU
Tasks are assigned dynamically to the CUs using a scheduling policy, but it does not take into account the computational power of the CUs.

Goal
Decide the best number of CUs (number of CPU cores and GPUs) to use for a given problem size $n$.

Tuning Technique
Selective Searching: successively adds CUs in increasing powerful order using as starting point the best value for $nb$ for the previous problem size $n$. 

Jesús Cámara (University of Murcia)
### Selection of Computing Units

#### Performance (GFlops)

<table>
<thead>
<tr>
<th>n</th>
<th>nb</th>
<th>CPU_Cores</th>
<th>GPU_IDs</th>
<th>Tuned</th>
<th>Chameleon</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>112</td>
<td>12</td>
<td>{−}</td>
<td>76</td>
<td>46</td>
<td>40</td>
</tr>
<tr>
<td>2000</td>
<td>192</td>
<td>9</td>
<td>{1,5,0}</td>
<td>164</td>
<td>117</td>
<td>29</td>
</tr>
<tr>
<td>3000</td>
<td>192</td>
<td>8</td>
<td>{1,5,0,2}</td>
<td>285</td>
<td>196</td>
<td>31</td>
</tr>
<tr>
<td>4000</td>
<td>240</td>
<td>7</td>
<td>{1,5,0,2,3}</td>
<td>412</td>
<td>352</td>
<td>15</td>
</tr>
<tr>
<td>5000</td>
<td>256</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>545</td>
<td>465</td>
<td>15</td>
</tr>
<tr>
<td>6000</td>
<td>256</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>626</td>
<td>554</td>
<td>12</td>
</tr>
<tr>
<td>7000</td>
<td>320</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>687</td>
<td>608</td>
<td>12</td>
</tr>
<tr>
<td>8000</td>
<td>320</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>753</td>
<td>682</td>
<td>10</td>
</tr>
<tr>
<td>9000</td>
<td>304</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>791</td>
<td>713</td>
<td>10</td>
</tr>
<tr>
<td>10000</td>
<td>304</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>834</td>
<td>758</td>
<td>9</td>
</tr>
<tr>
<td>11000</td>
<td>512</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>880</td>
<td>803</td>
<td>9</td>
</tr>
<tr>
<td>12000</td>
<td>576</td>
<td>6</td>
<td>{1,5,0,2,3,4}</td>
<td>909</td>
<td>896</td>
<td>1</td>
</tr>
</tbody>
</table>
Overview

1 Motivation

2 Application Case

3 Search of the Block Size

4 Selection of Computing Units

5 Conclusions and Future Work
Conclusions

- The *pruned* strategy allows us to achieve good results both in the *empirical* and in the *simulated* approaches.

- Results obtained with the *simulated* approach are very similar in terms of performance to that achieved by the *empirical* one.

- The combination *Simulated+Pruned* is a decent tuning strategy to obtain performances close to the *empirical* without (almost) requiring to use the computational platform.

- An appropriate selection of the CUs allows to improve the performance with an efficient exploitation of the heterogeneous platform.
Future Work

- Apply the empirical and simulated tuning strategies to other routines of Chameleon, such as LU or QR.

- Consider additional parameters: inner block ($ib$) and scheduling policies ($eager$, $lws$, $dmda$, . . .)

- Integrate the autotuning techniques inside Chameleon.
Thanks for your Attention!