Servet: A Benchmark Suite for Autotuning on Multicore Clusters

Jorge González-Domínguez*, Guillermo L. Taboada, Basilio B. Fraguela, María J. Martín, Juan Touriño

Computer Architecture Group
University of A Coruña (Spain)
{jgonzalezd,taboada,basilio.fraguela, mariam,juan}@udc.es

24th IEEE International Parallel and Distributed Processing Symposium (IPDPS’10), Atlanta, GE, USA
1. Introduction
   - Autotuned Codes
   - Extraction of System Parameters

2. Cache Topology
   - Cache Size Estimate
   - Determination of Shared Caches

3. Memory Access Overhead Characterization

4. Determination of Communication Costs

5. Conclusions
1. Introduction
   - Autotuned Codes
   - Extraction of System Parameters

2. Cache Topology

3. Memory Access Overhead Characterization

4. Determination of Communication Costs

5. Conclusions
Autotuning

Codes that can automatically adapt their performance to the machine on what they are executed.

Autotuned Sequential Libraries

- **ATLAS** -> Numerical computing (BLAS)
- **FFTW3** -> Discrete Fourier transform
- **Spiral** -> Digital signal processing (DSP) algorithms

Wide search mechanism to find the most suitable algorithm

The knowledge of some hardware characteristics can reduce their search times
Examples of Autotuning Techniques

- **Tiling** -> To divide the computation in blocks of data which fit in cache to minimize the number of cache misses.

- **Efficient communications:**
  - Minimizing the use of interconnection networks
  - Increasing the use of shared memory -> usually faster
Examples of Autotuning Techniques

- **Tiling** -> To divide the computation in blocks of data which fit in cache to minimize the number of cache misses.
- **Efficient communications:**
  - Minimizing the use of interconnection networks
  - Increasing the use of shared memory -> usually faster
Autotuning Techniques (II)

Node 0

Node 1

Shared Memory

11 10 9 8

12 13 14 15

16 17 18 19

20 21 22 23

24 25 26 27

28 29 30 31

Interconnection Network
Communications Algorithm: Option 1 (I)
Communications Algorithm: Option 1 (II)

Node 0

Node 1

Interconnection Network
Communications Algorithm: Option 1 (III)
Communications Algorithm: Option 1 (IV)
Communications Algorithm: Option 1 (and V)

Node 0

Node 1

Interconnection Network
Communications Algorithm: Option 2 (II)

Node 0

Node 1

Interconnection Network
Communications Algorithm: Option 2 (III)
Communications Algorithm: Option 2 (and IV)
Examples of Autotuning Techniques

- **Tiling** -> To divide the computation in blocks of data which fit in cache to minimize the number of cache misses.
- **Efficient communications:**
  - Minimizing the use of interconnection networks
  - Increasing the use of shared memory -> usually faster
- **Mapping policies to minimize:**
  - Cache misses because of shared caches
  - Memory access overheads
  - Use of interconnection networks
Mapping Policies: Reducing Communication Costs

Node 0

Node 1

Interconnection Network
Mapping Policies: Reducing Memory Access Overhead

Node 0

Node 1

Interconnection Network
Obtaining the System Parameters

Option 1 -> From the machine specifications
- Always?
- Where?
- Restricted?
- What format?
- Accurate?

Option 2 -> With benchmarks
- General -> Can be used always without restrictions
- Portable -> The place and format do not depend on the vendor
- Accurate -> Explore the real behavior of the machine
Obtaining the System Parameters

**Option 1 -> From the machine specifications**
- Always?
- Where?
- Restricted?
- What format?
- Accurate?

**Option 2 -> With benchmarks**
- General -> Can be used always without restrictions
- Portable -> The place and format do not depend on the vendor
- Accurate -> Explore the real behavior of the machine
Lacks of Previous Works

- Non portable method to estimate sizes of physically indexed caches
- Do not consider different memory access overheads
- Poor communication characterization -> Not necessary in multicores
- Code not available
1. Introduction

2. Cache Topology
   - Cache Size Estimate
   - Determination of Shared Caches

3. Memory Access Overhead Characterization

4. Determination of Communication Costs

5. Conclusions
Main Idea

$T \rightarrow$ Average Access Time

$T_A < T_B < T_C$
Main Idea

\[ T \rightarrow \text{Average Access Time} \]

\[ T_A < T_B < T_C \]
Dunnington Example: Cycles View

The graph shows the average cycles per access as a function of array size (in bytes). The x-axis represents different array sizes ranging from 4K to 13M bytes. The y-axis represents the average cycles per access, ranging from 10 to 1000 cycles. The line indicates the performance trend for the Dunnington-E7450 model.
Dunnington Example: Cycles View

![Graph of Average Cycles per Access vs. Array Size (bytes) for Dunnington-E 7450](image-url)
Dunnington Example: Gradient View (I)

$L1 = 32KB \sqrt{\text{.}}$
Dunnington Example: Gradient View (I)

\[ L_1 = 32\, KB \checkmark \]
\[ L_2 = 1\, MB \times \]
L2 Problem

Physically Indexed Caches

- Most of L2 and L3 caches
- If cache size larger than page size contiguity in virtual memory does not imply adjacency in physical memory
- Cache misses in tests with array sizes smaller than the cache considered

Solutions

- Working as virtually indexed caches
  - Page Coloring by the OS -> Not in Linux
  - Calls to OS functions -> Previous works -> Not portable
- Estimating from the physically indexed behavior -> Servet
L2 Problem

Physically Indexed Caches

- Most of L2 and L3 caches
- If cache size larger than page size contiguity in virtual memory does not imply adjacency in physical memory
- Cache misses in tests with array sizes smaller than the cache considered

Solutions

- Working as virtually indexed caches
  - Page Coloring by the OS -> Not in Linux
  - Calls to OS functions -> Previous works -> Not portable
- Estimating from the physically indexed behavior -> Servet
Probabilistic Algorithm (I)

Statistics Aspects of Cache Misses

- Page Size $\rightarrow$ $PS$
- Cache: Size $\rightarrow$ $CS$; Associativity $\rightarrow$ $K$; number of Page Sets $\rightarrow$ $CS/(K \ast PS)$
- Number of Pages in a test $\rightarrow$ $NP$
- Probability of a given virtual page is mapped to a given page set is uniform $\Rightarrow$ Number of pages $X$ per page set $\in B(NP, (K \ast PS)/CS)$
- As each set can contain up to $K$ pages without conflict $\Rightarrow (X > K)$ is the miss rate when accessing to $NP$ pages
Probabilistic Algorithm (and II)

Algorithm

Entries: $S[n], C[n]$

$hit\_time = \text{MIN}(C)$; $miss\_overhead = \text{MAX}(C) - \text{MIN}(C)$

$for\ (i = 0; i < n; i = i + 1)$

$MR[i] = (C[i] - hit\_time)/miss\_overhead$

$NP[i] = S[i]/PS$

$foreach(CS, K)$

$div[CS][K] = 0$

$for\ (i = 0; i < n; i = i + 1)$

$\qquad div[CS][K] = div[CS][K] + | MR[i] - P(X > K) |$

$\quad X \in B(NP[i], (K \times PS)/CS)$

Result: The statistical mode of $CS$ using the five elements of $div$ with the lowest values
Dunnington Example: Gradient View (and II)

\[ L_1 = 32KB \checkmark \]
\[ L_2 = 3MB \checkmark \]
Dunnington Example: Gradient View (and II)

\begin{equation}
L_1 = 32\, KB \checkmark \quad L_2 = 3\, MB \checkmark \quad L_3 = 12\, MB \checkmark
\end{equation}
Experimental Evaluation

Academic Environment

- 70 different machines
- 147 different caches
- 140 correct estimations -> 95%
- Solving the bad estimations -> Next version of Servet
Main Idea

1. Not Shared Cache

2. Shared Cache

\[ T \rightarrow \text{Average Access Time} \]

\[ T_1 < T_2 \]
Determination of Shared Caches

Algorithm

```plaintext
foreach(CS)
    ref = number of cycles to access only one core to an array of size (CS * 2)/3
    foreach(pair of cores)
        c = number of cycles to access both cores simultaneously to an array of size (CS * 2)/3
        ratio = c/ref
        if ratio > 2 then SHARED CACHE
```
Initial Topology
Dunnington Example: L1 Results

\[ L1 \rightarrow \text{not shared} \]
Topology with L1
Dunnington Example: L2 Results

$L2 \rightarrow \text{shared by 0 and 12}$
Topology with L2
Dunnington Example: L3 Results

L3 → shared by 0,1,2,12,13 and 14
Topology with L3

- L1
- L2
- L3

Levels and Numbers:
- L1: 0, 1, 2, 3, 4, 5
- L2: 12, 13, 14, 15, 16
- L3: 18, 19, 20, 21, 22, 23
Introduction

Cache Topology

Memory Access Overhead Characterization

Determination of Communication Costs

Conclusions
Detection of Different Memory Access Overheads

Algorithm

\[ n = 0 \]
\[ \text{ref} = \text{memory bandwidth when accessing one isolated core} \]
\[ \text{foreach}(\text{pair of cores}) \]
\[ b = \text{bandwidth of one process when accessing both cores} \]
\[ \text{if}(b < \text{ref}) \]
\[ \quad \text{if}(b \text{ similar to a given } BW[i]) \]
\[ \quad \text{Add the pair to } P_m[i] \]
\[ \text{else} \]
\[ BW[n] = b \]
\[ P_m[n] = \text{The used pair} \]
\[ n = n + 1 \]
60% of bandwidth when accessing by pairs
Finis Terrae Architecture (I)
Finis Terrae Architecture (II)

Cell 0

1 NODE -> 2 CELLS
1 CELL -> 4 PROCESSORS
1 PROCESSOR -> 2 CORES
Finis Terrae Architecture (and III)

1 NODE -> 2 CELLS
1 CELL -> 4 PROCESSORS
1 PROCESSOR -> 2 CORES

SHARED MEMORY -> 8 CORES
SHARED BUS -> 4 CORES
Finis Terrae Results (I)
Finis Terrae Results (and II)

Memory Bandwidth in Finis Terrae

- Isolated accesses
  - 2200 MBytes/s

- Cores with the same bus
  - 990 MBytes/s
  - Only 45% of bandwidth

- Cores in the same cell with different bus
  - 1650 MBytes/s
  - 75% of bandwidth

- Cores in different cell
  - The same bandwidth
Memory Access Bandwidth per Overhead

- **Dunnington**
- **Finis Terrae (bus)**
- **Finis Terrae (cell)**

![Graph showing bandwidth vs. number of cores accessing memory concurrently](image-url)
Algorithm

\[ n = 0 \]

foreach (pair of cores)

\[ l = \text{latency with a message of L1 size between the two cores} \]

if \( b \) similar to a given \( L[i] \)

Add the pair to \( P_i[i] \)

else

\[ L[n] = l \]

\[ P_i[n] = \text{The used pair} \]

\[ n = n + 1 \]
Dunnington Results (I)

![Graph showing latency in microseconds for core receiving messages from core 0.](image-url)
Dunnington Results (I)
Dunnington Results (I)
Dunnington Results (I)

Graph showing latency (microseconds) vs. core number, with peaks at certain core numbers and a line labeled "Dunnington."
Dunnington Results (and II)

Dunnington (messages of L1 size)

- Intra-processor communications (sharing L2 cache)
  - 2130 MBytes/s
- Intra-processor communications (not sharing L2 cache)
  - 1780 MBytes/s
  - 83% of possible the highest bandwidth
- Inter-processor communications
  - 750 MBytes/s
  - Only 35% of possible highest bandwidth
Characterization of Communication Layers

![Graph showing bandwidth vs. message size for different communication layers.](image)
1 Introduction

2 Cache Topology

3 Memory Access Overhead Characterization

4 Determination of Communication Costs

5 Conclusions
Suite to detect hardware parameters:
- Cache sizes
- Shared caches topology
- Memory accesses overheads
- Communications bandwidths

Characteristics
- Portable
- Highly accurate
- Focused to support the autotuning on multicore clusters
- Freely available: http://servet.des.udc.es
HAVE A TRY AND ENJOY!!!

http://servet.des.udc.es

Contact: Jorge González-Domínguez  
jgonzalezd@udc.es
Computer Architecture Group,
Dept. of Electronics and Systems,
University of A Coruña, Spain