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

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

Autotuning Techniques (I)

Examples of Autotuning Techniques

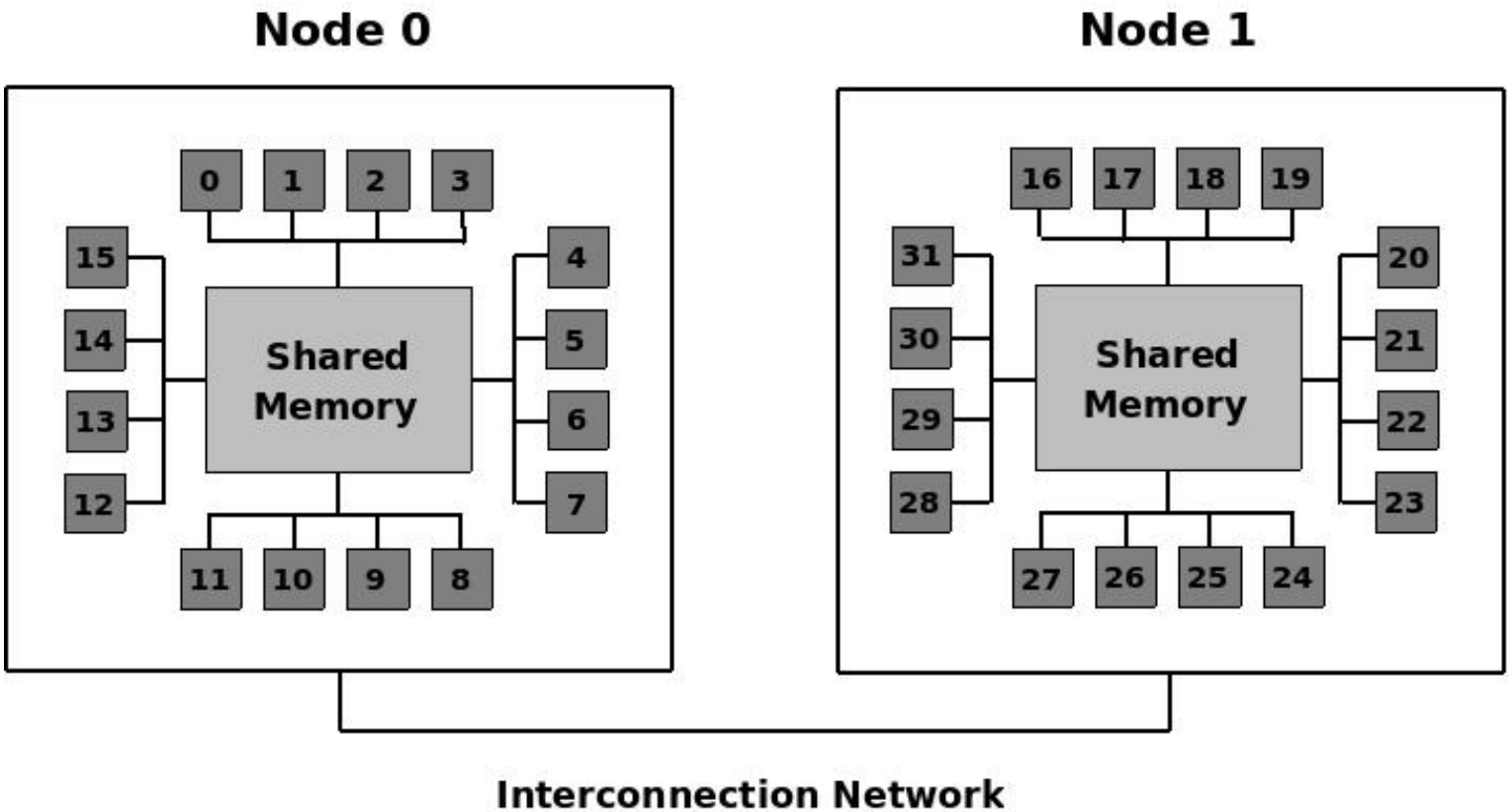
- Tiling -> To divide the computation in blocks of data which fit in cache to minimize the number of cache misses.
- Efficient communications:
  - Minimizing the use of interconnection networks
  - Increasing the use of shared memory -> usually faster

Autotuning Techniques (I)

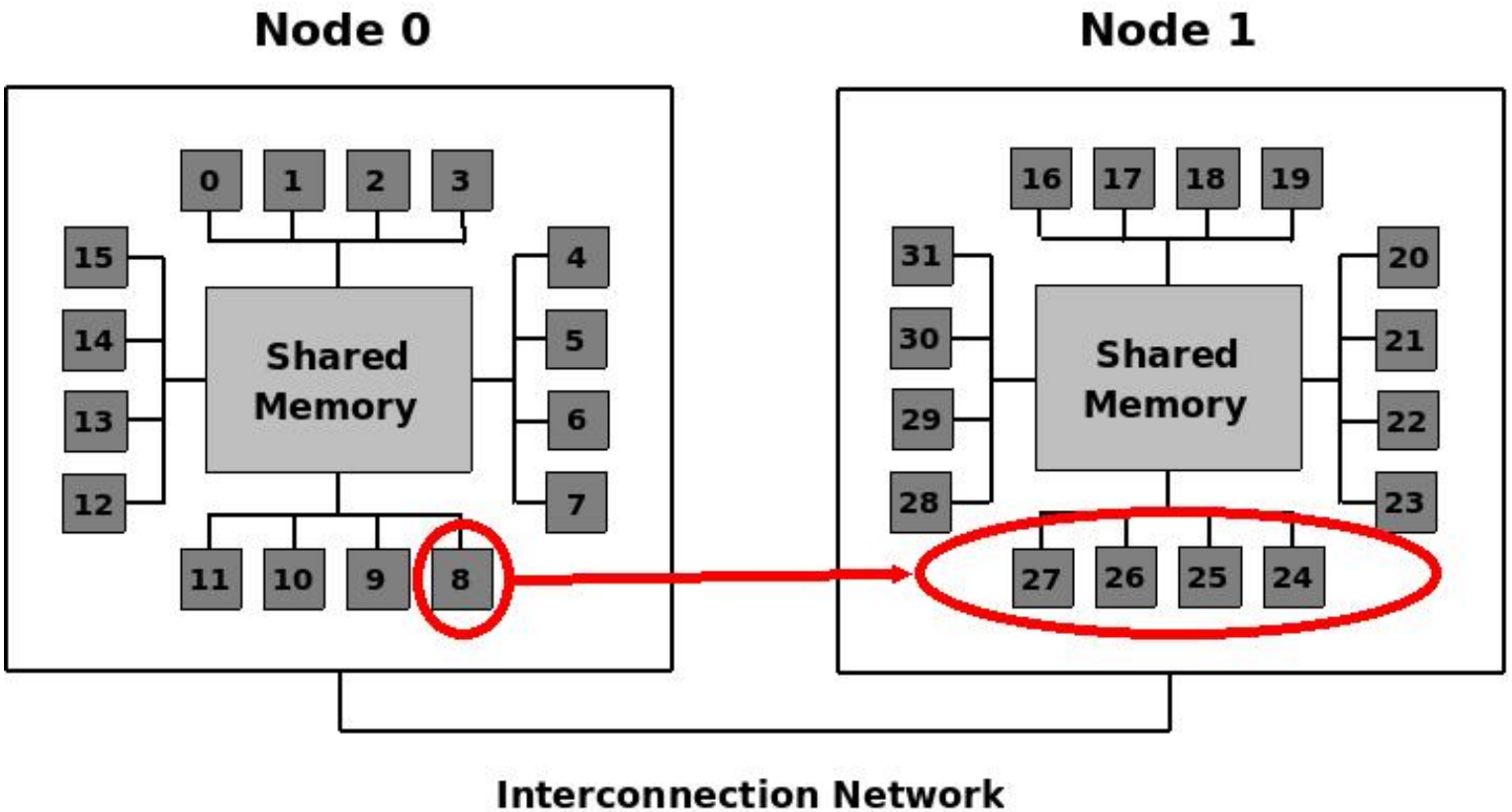
Examples of Autotuning Techniques

- 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

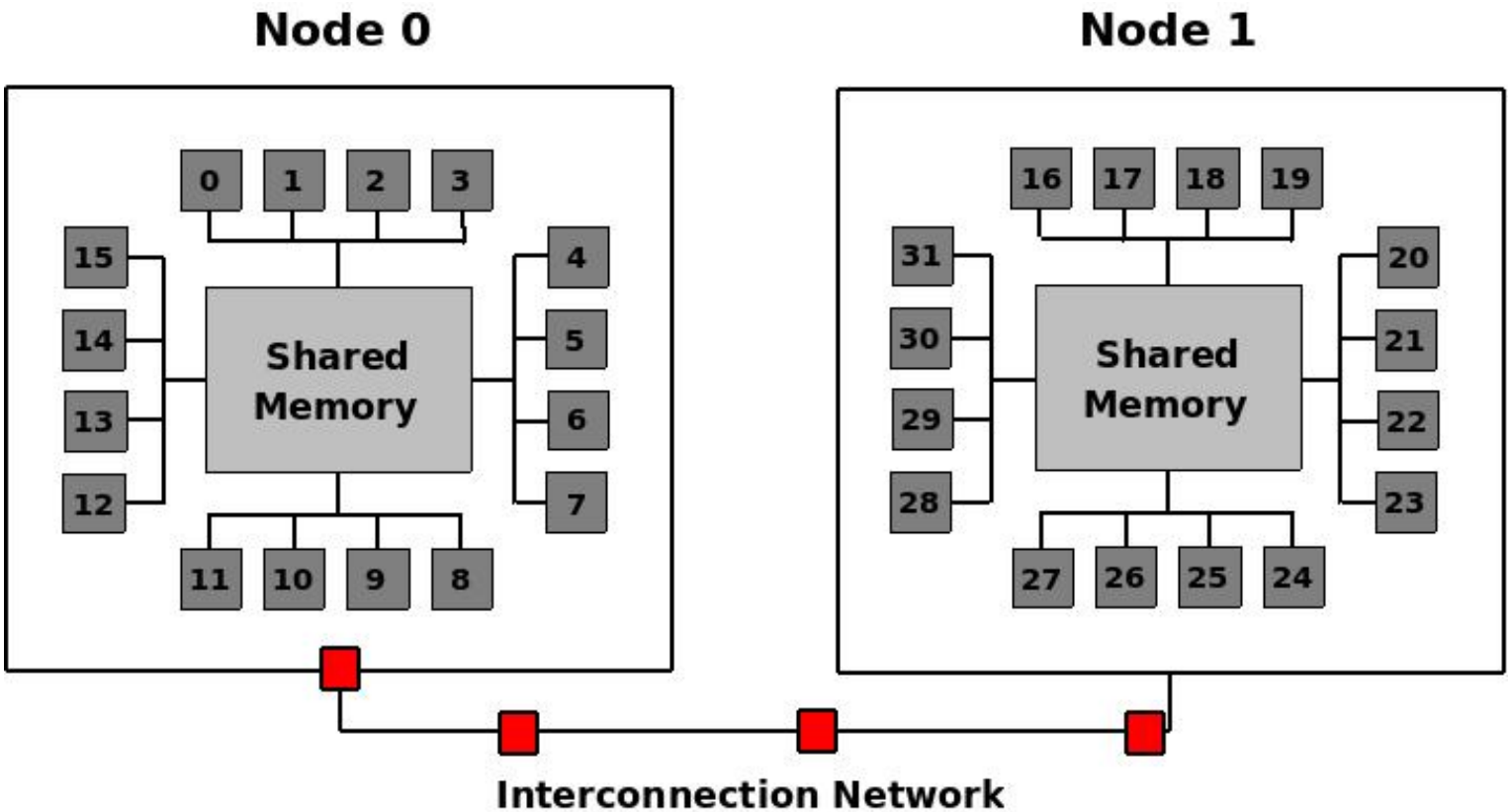
Autotuning Techniques (II)



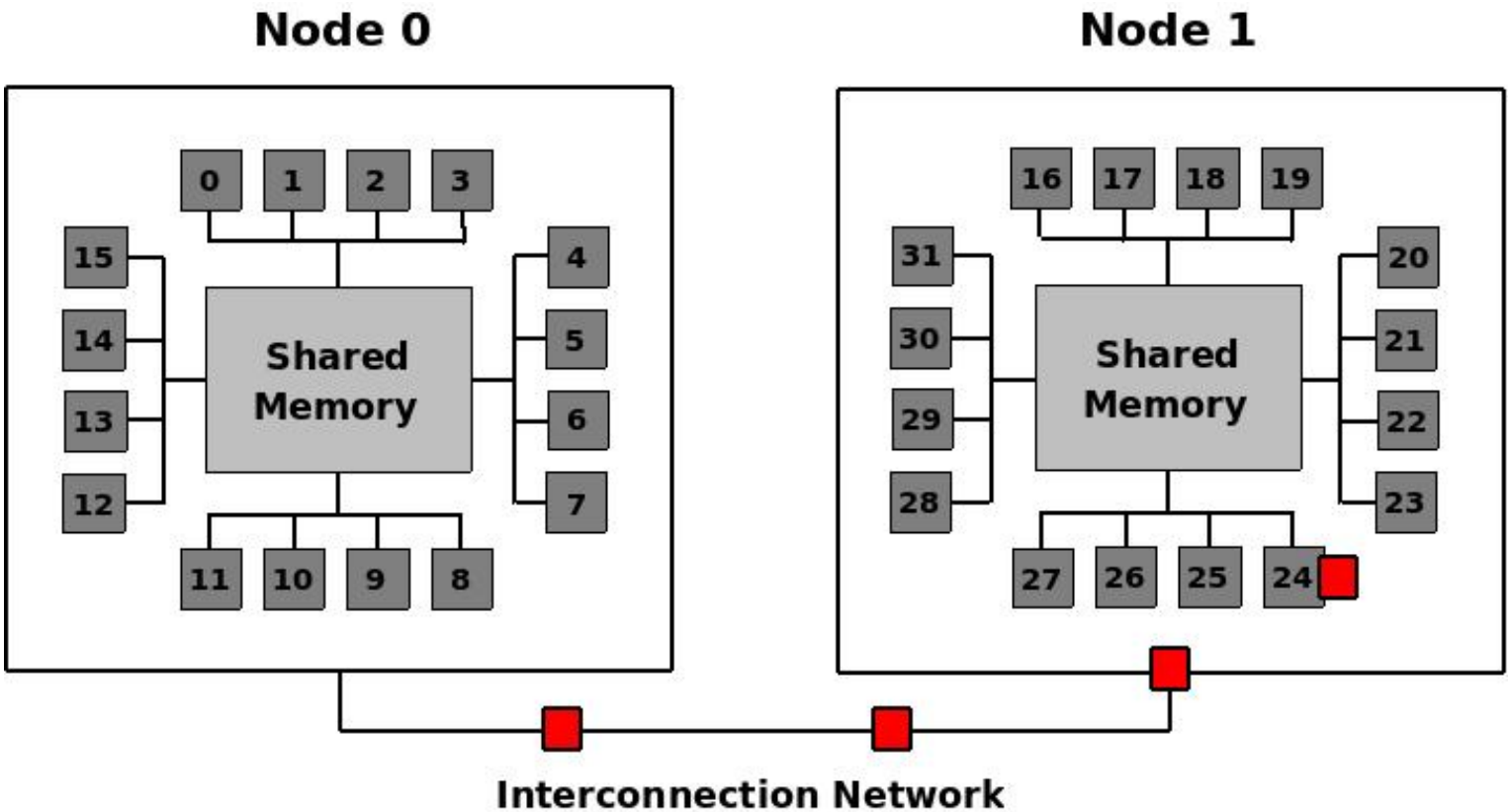
Communications Algorithm: Option 1 (I)



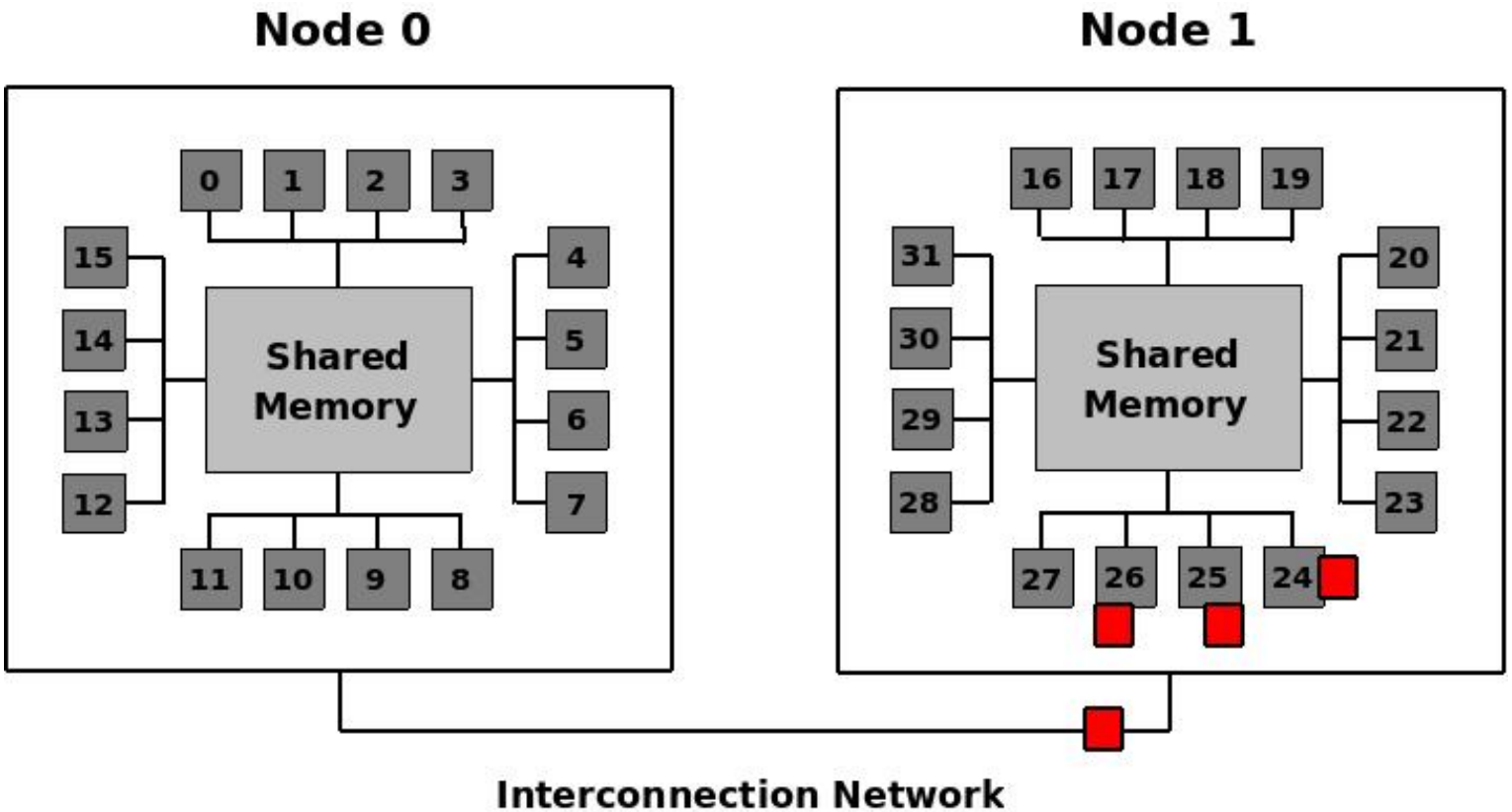
Communications Algorithm: Option 1 (II)



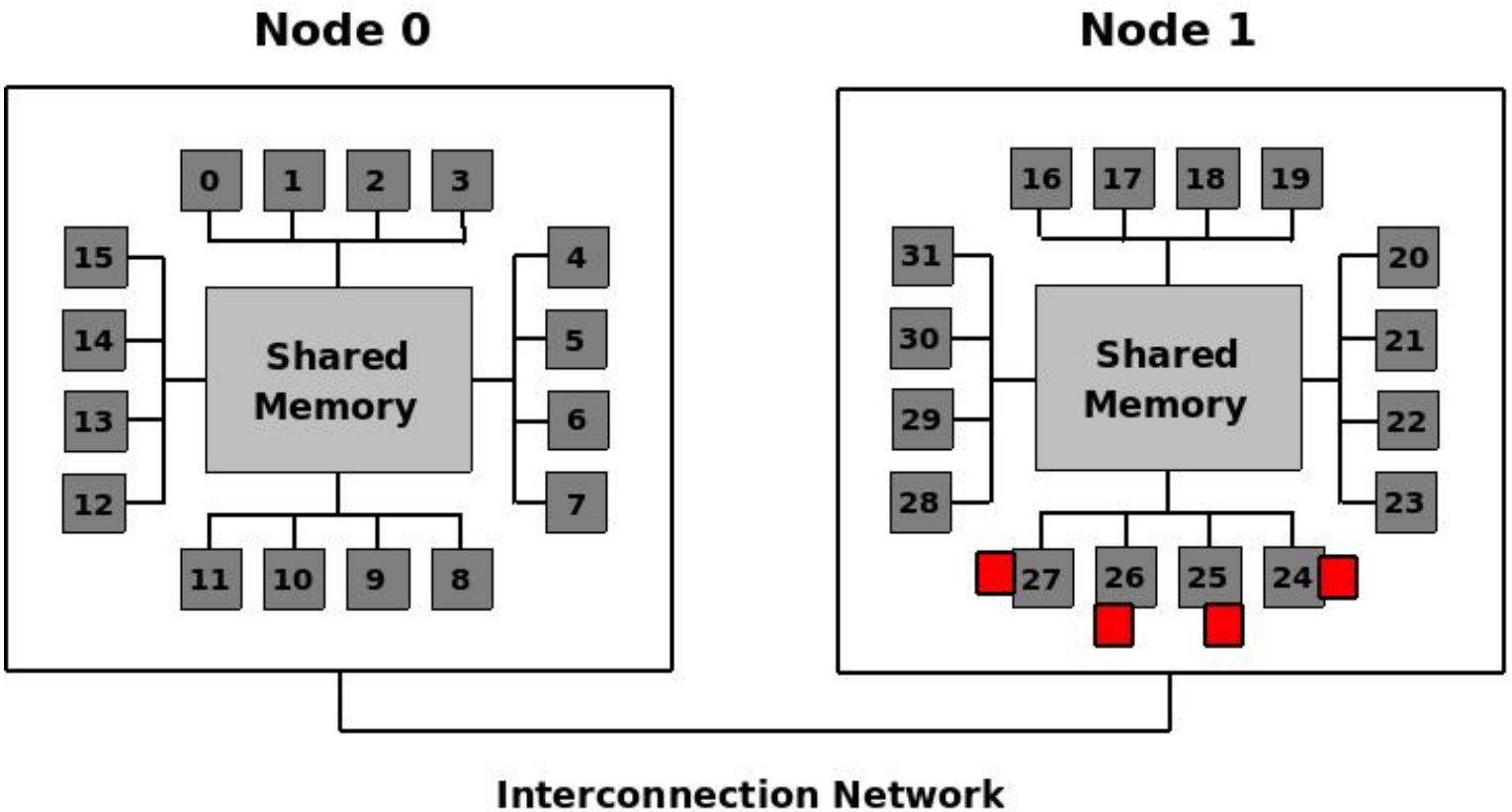
Communications Algorithm: Option 1 (III)



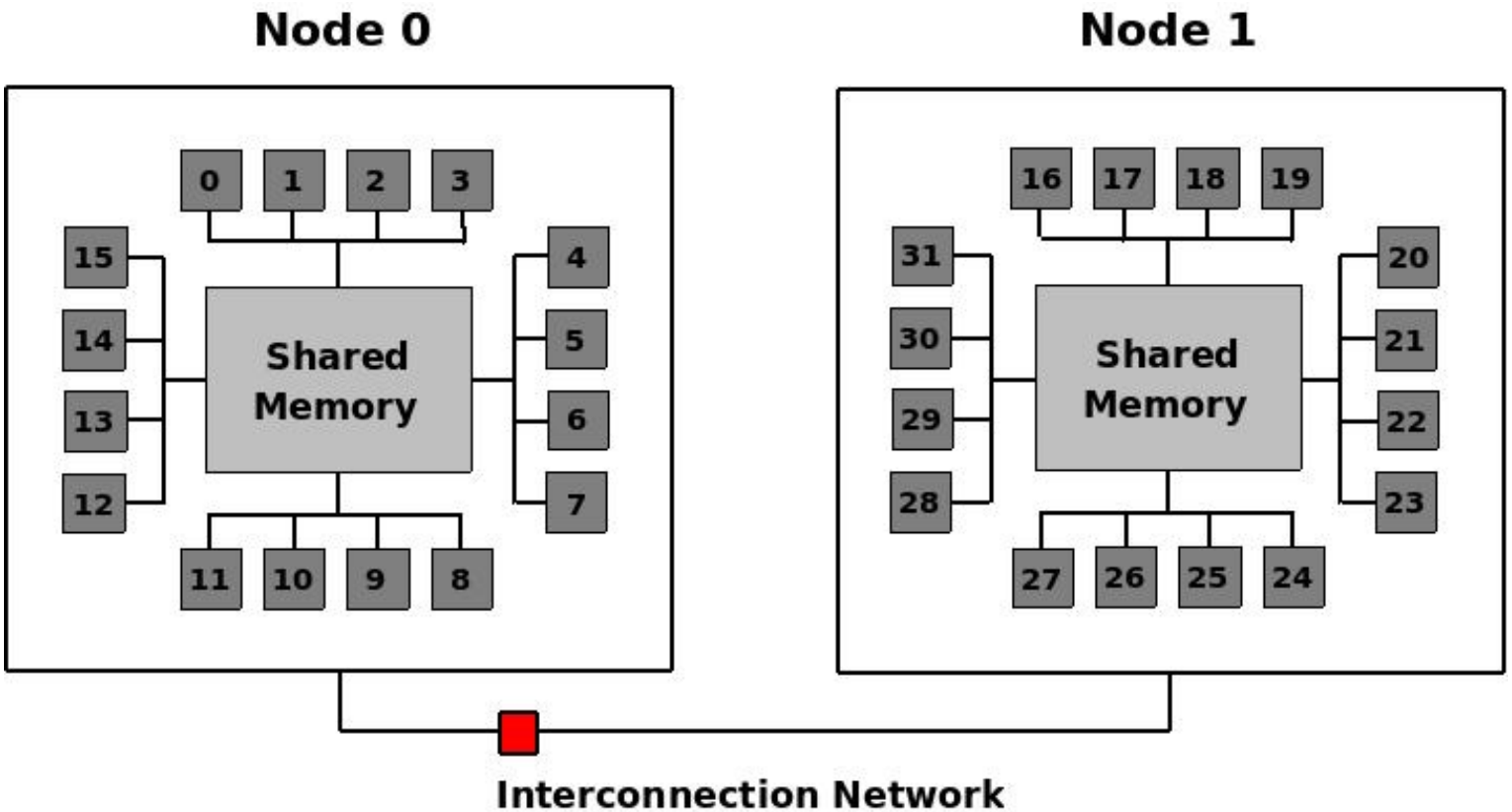
Communications Algorithm: Option 1 (IV)



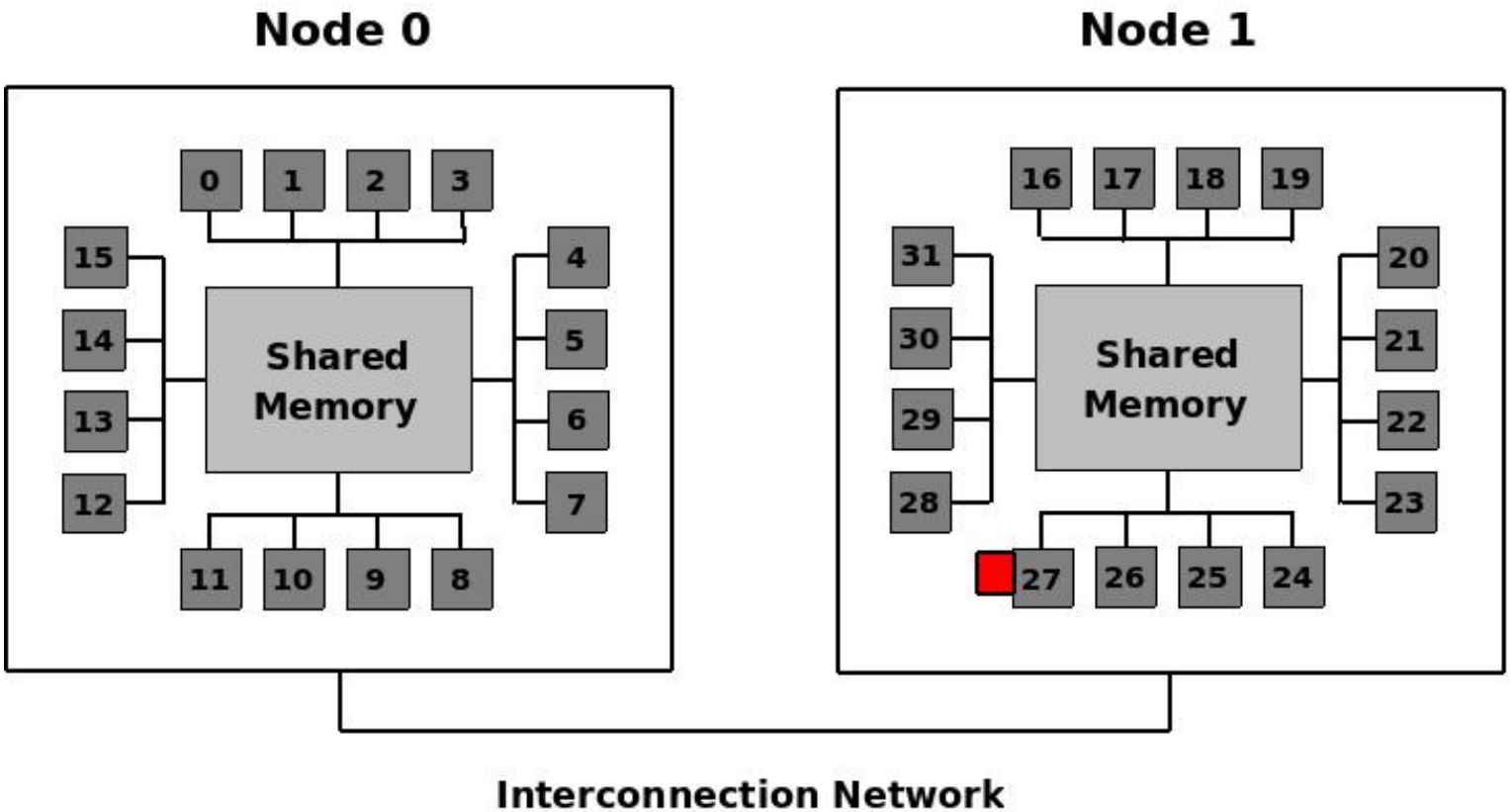
Communications Algorithm: Option 1 (and V)



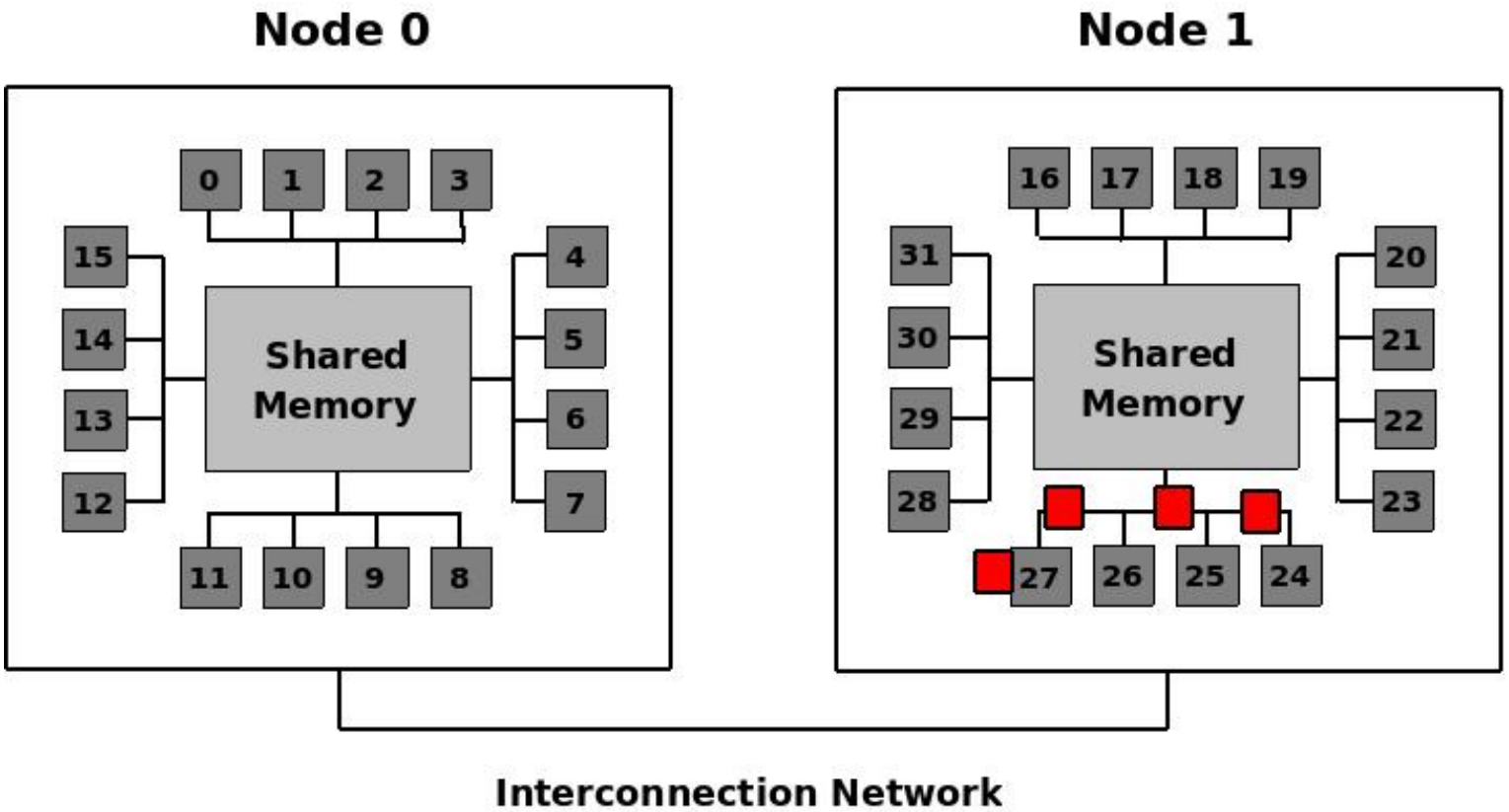
Communications Algorithm: Option 2 (I)



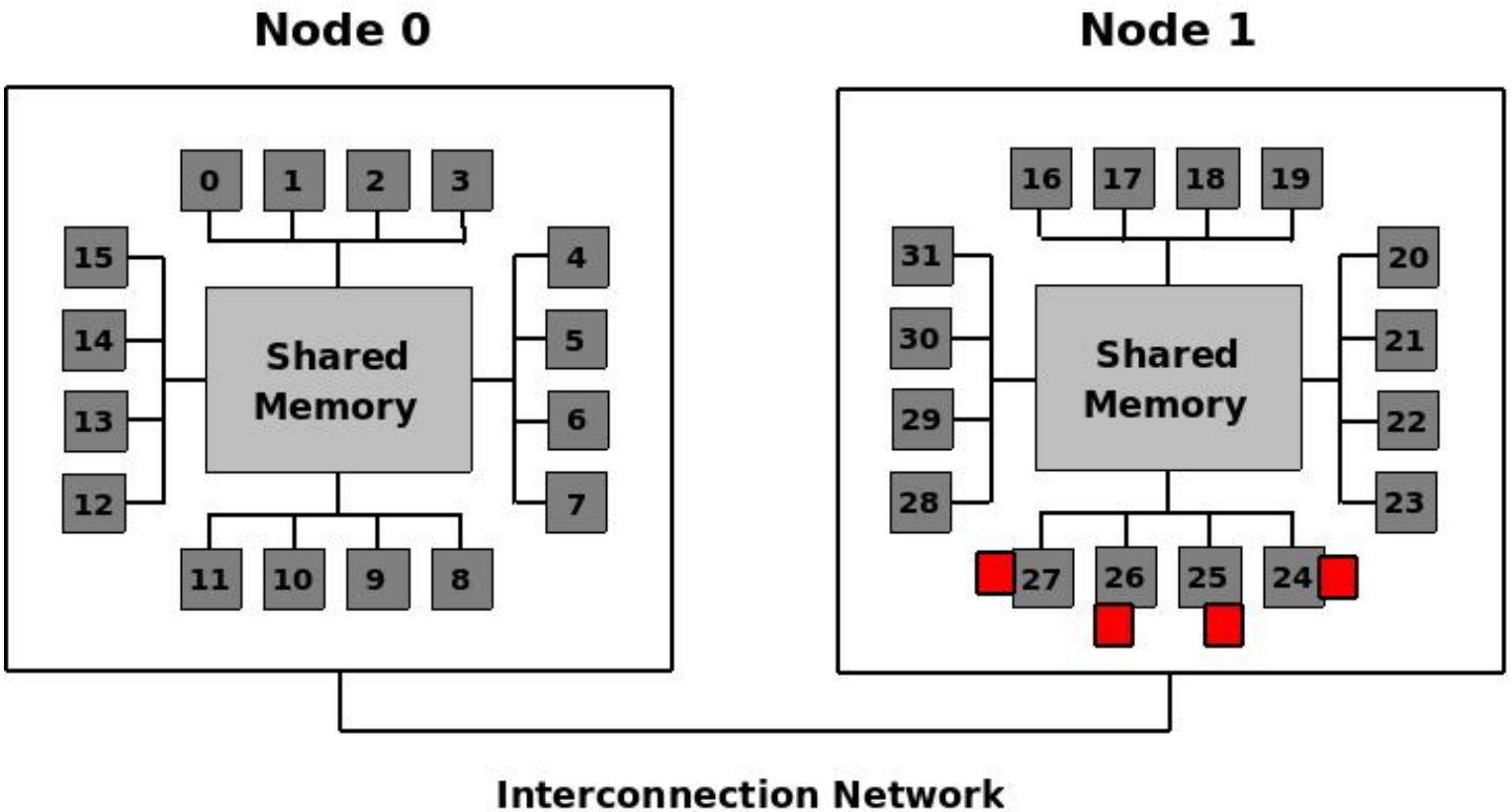
Communications Algorithm: Option 2 (II)



Communications Algorithm: Option 2 (III)



Communications Algorithm: Option 2 (and IV)

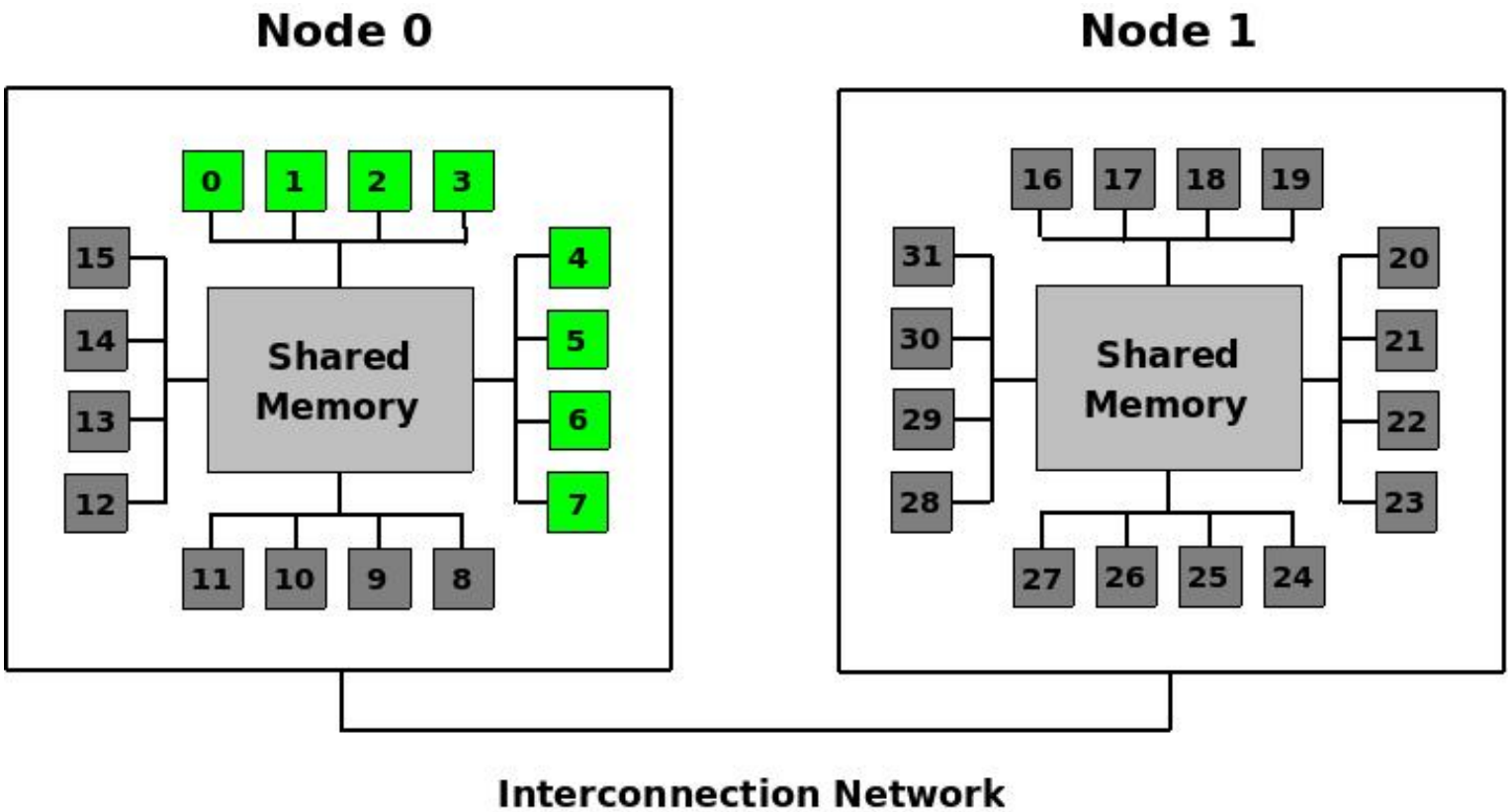


Autotuning Techniques (and III)

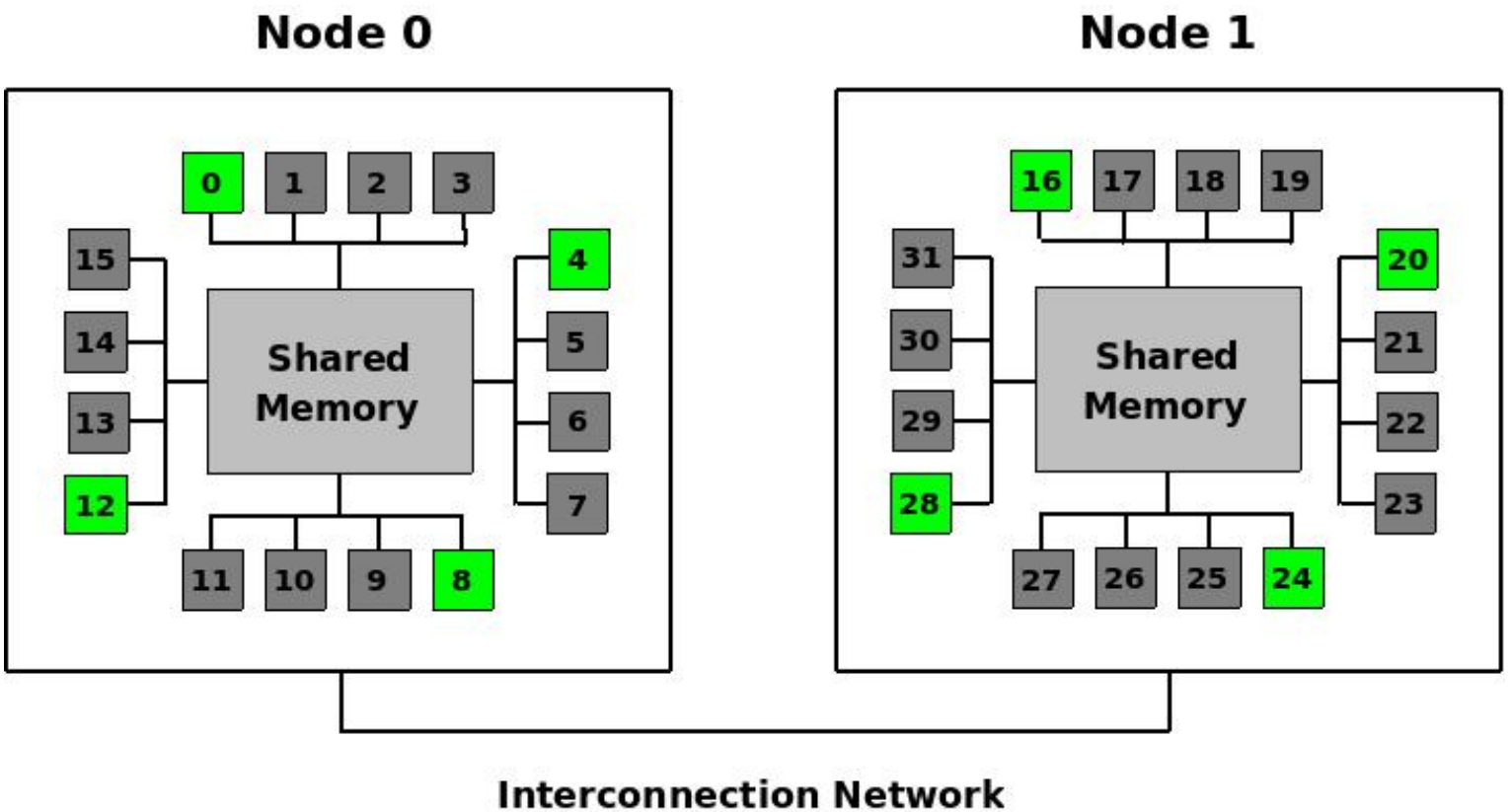
Examples of Autotuning Techniques

- 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



Mapping Policies: Reducing Memory Access Overhead



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

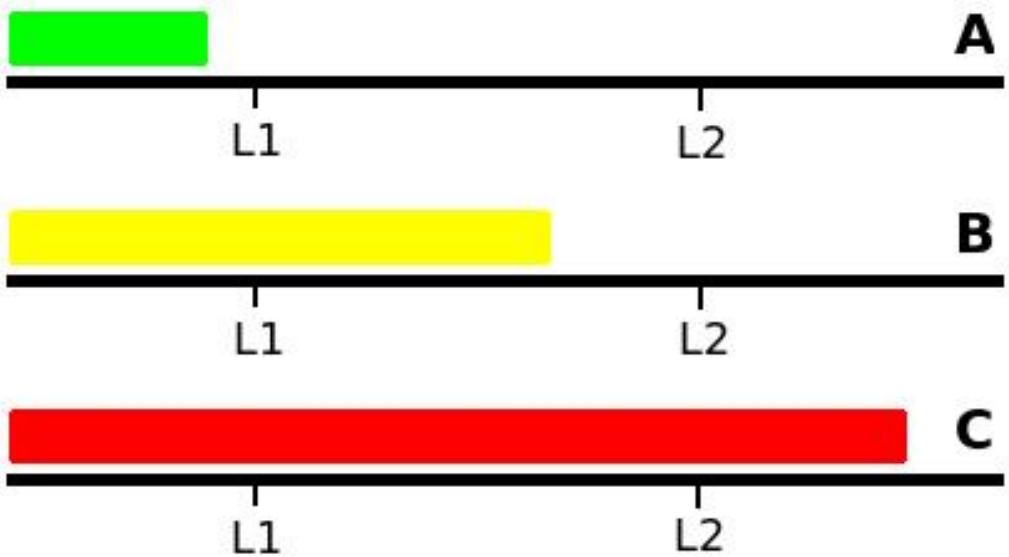
- General -> Can be used always without restrictions
- Portable -> The place and format do not depend on the vendor
- Accurate -> Explore the real behavior of the machine

Related Work

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

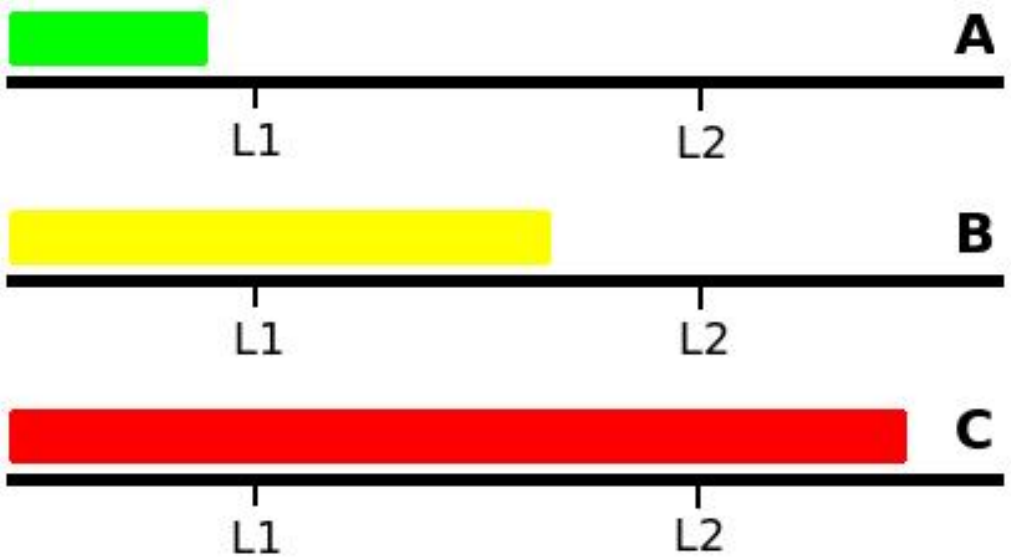
Main Idea



T -> Average Access Time

$T_A < T_B < T_C$

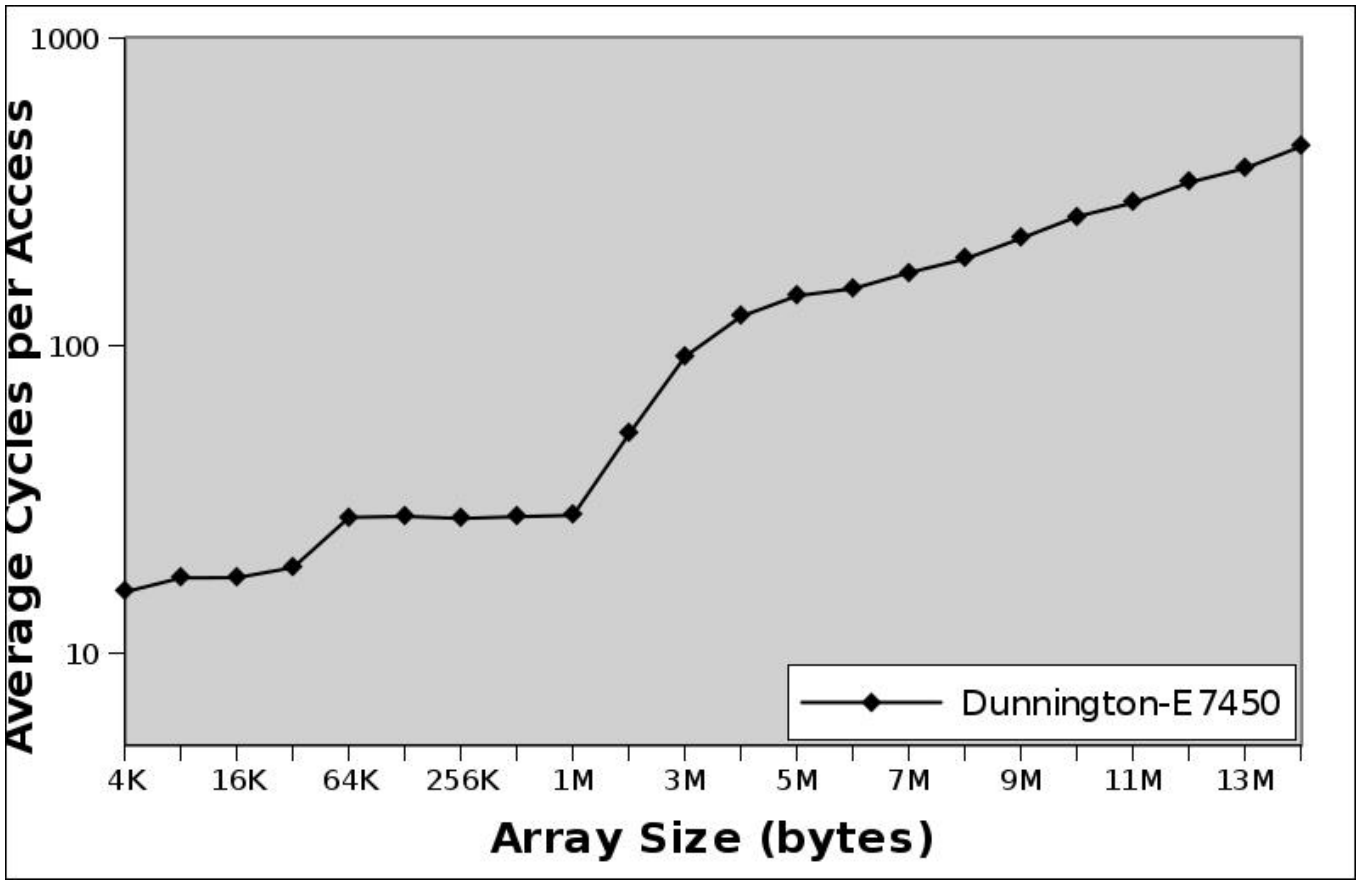
Main Idea



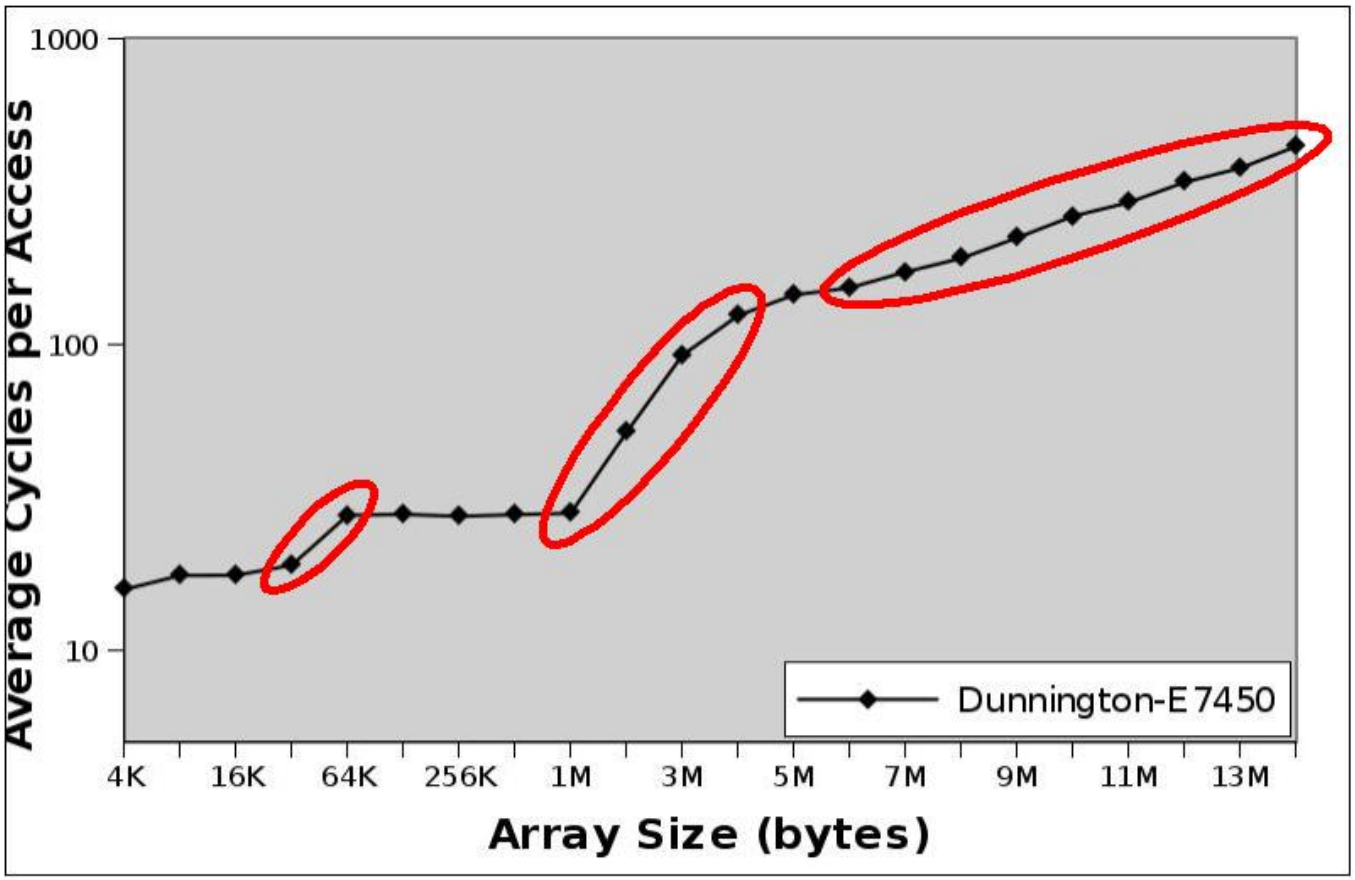
T -> Average Access Time

$T_A < T_B < T_C$

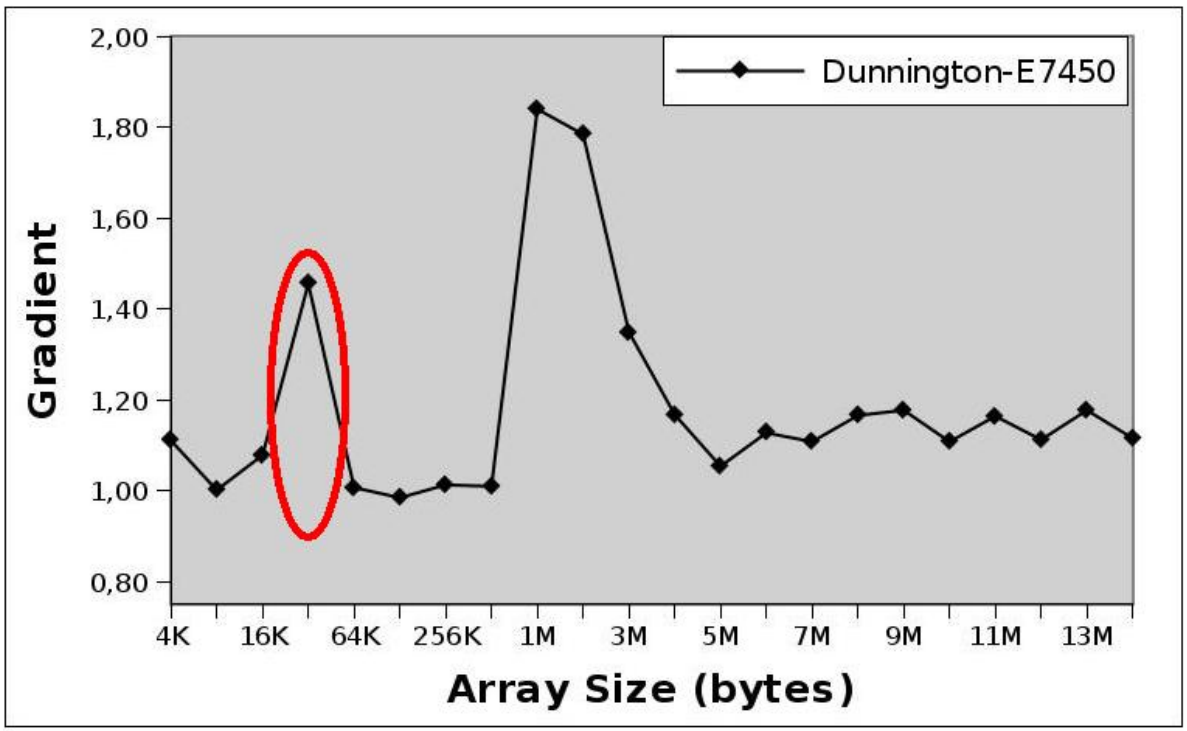
Dunnington Example: Cycles View



Dunnington Example: Cycles View

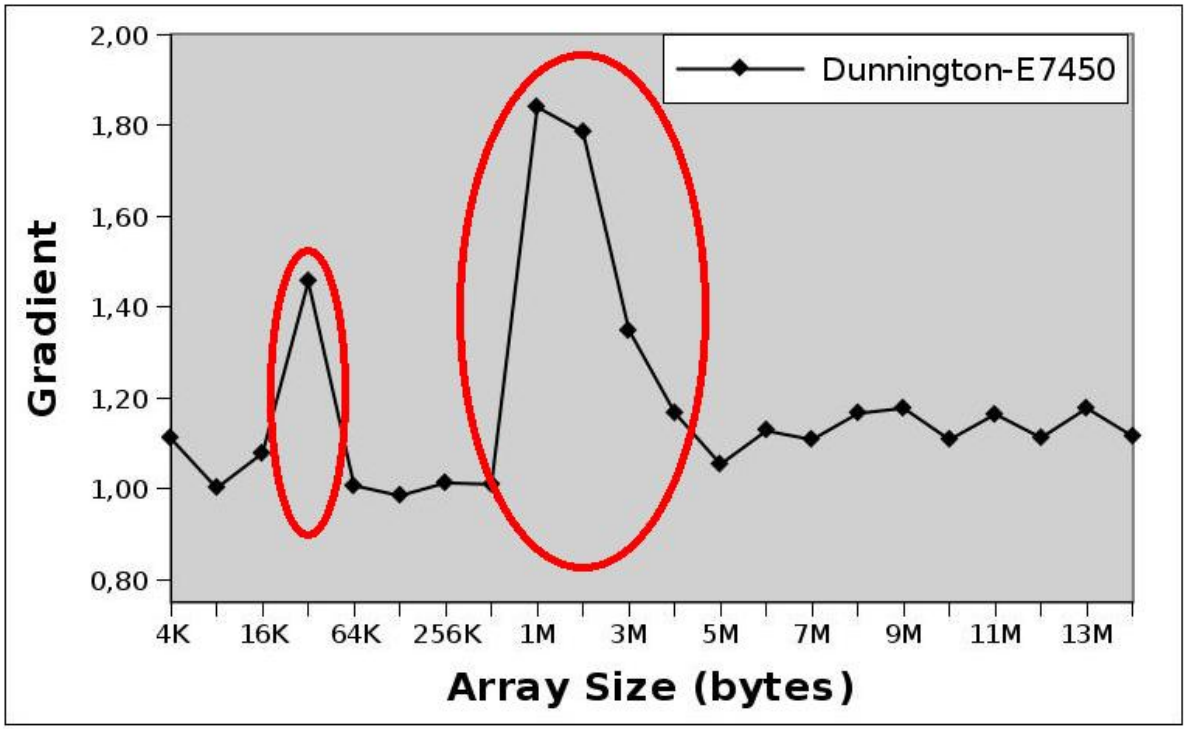


Dunnington Example: Gradient View (I)



$$L1 = 32KB\checkmark$$

Dunnington Example: Gradient View (I)



$$L1 = 32KB\checkmark$$
$$L2 = 1MB\ \text{X}$$

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

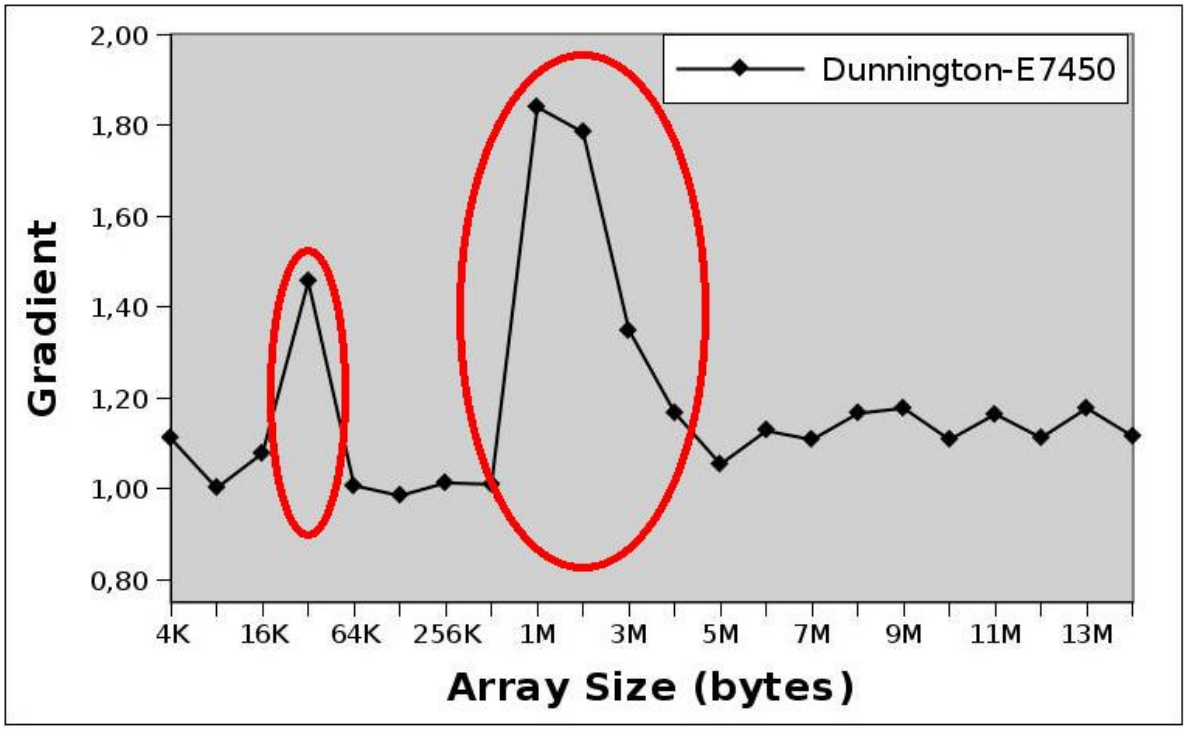
- Page Size -> PS
- Cache: Size -> CS; Associativity -> K; number of Page Sets -> $CS/(K * PS)$
- Number of Pages in a test -> NP
- Probability of a given virtual page is mapped to a given page set is uniform $\Rightarrow$ Number of pages X per page set $\in B(NP, (K * PS)/CS)$
- As each set can contain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

Probabilistic Algorithm (and II)

Algorithm

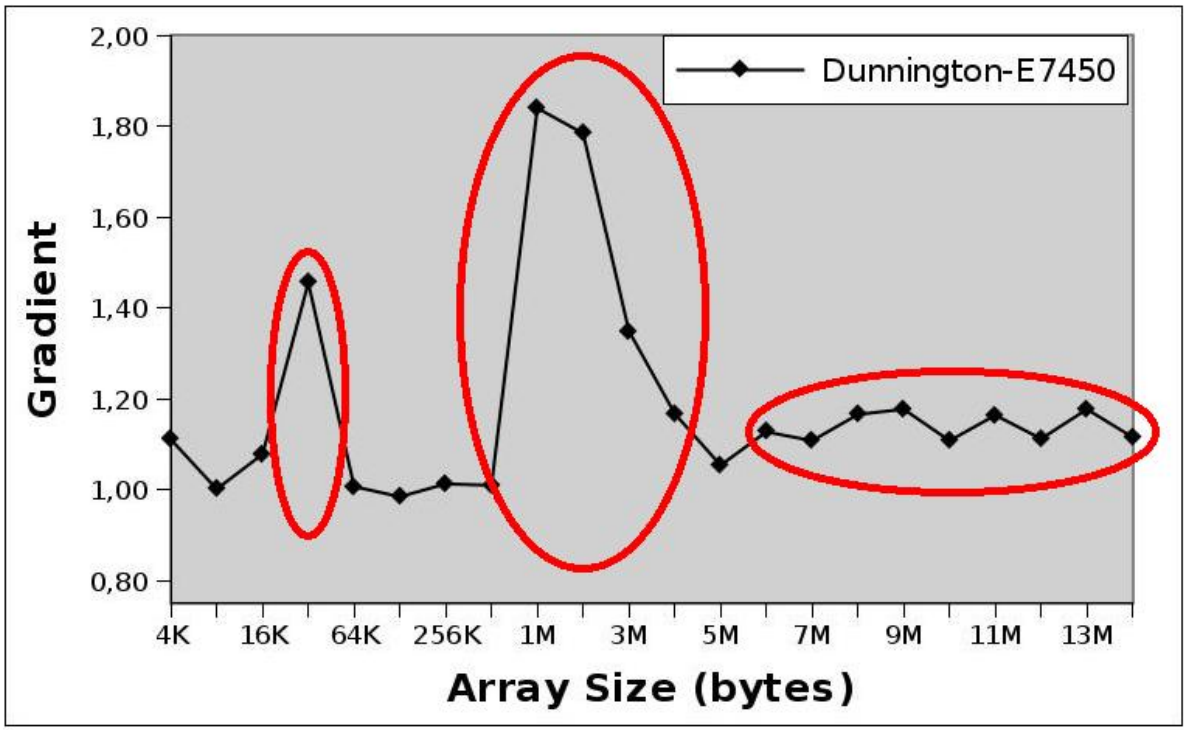
Entries: $S[n], C[n]$
$hit\_time = MIN(C); miss\_overhead = MAX(C) - MIN(C)$
$for(i = 0; i < n; i = i + 1)$
    $MR[i] = (C[i] - hit\_time)/miss\_overhead$
    $NP[i] = S[i]/PS$
$foreach(CS, K)$
    $div[CS][K] = 0$
    $for(i = 0; i < n; i = i + 1)$
        $div[CS][K] = div[CS][K] + \mid MR[i] - P(X > K) \mid$
                $X \in B(NP[i], (K * PS)/CS)$
Result: The statistical mode of CS using the five elements of div with the lowest values

Dunnington Example: Gradient View (and II)



$$L1 = 32KB\checkmark$$
$$L2 = 3MB\checkmark$$

Dunnington Example: Gradient View (and II)



$$L1 = 32KB\checkmark$$
$$L2 = 3MB\checkmark$$
$$L3 = 12MB\checkmark$$

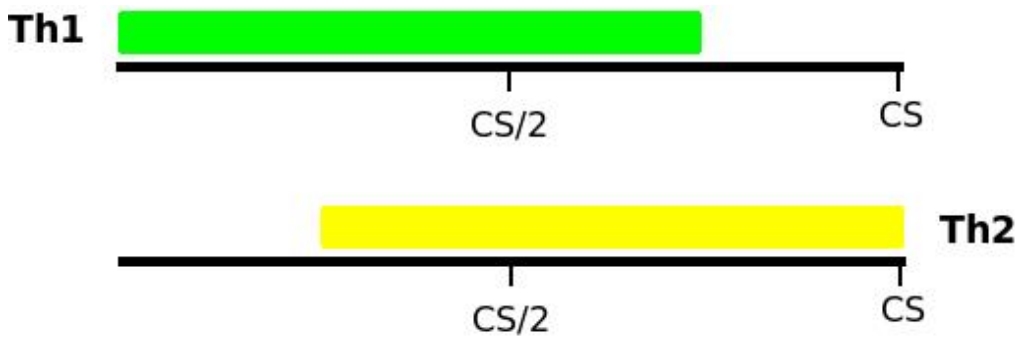
Experimental Evaluation

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 -> Average Access Time

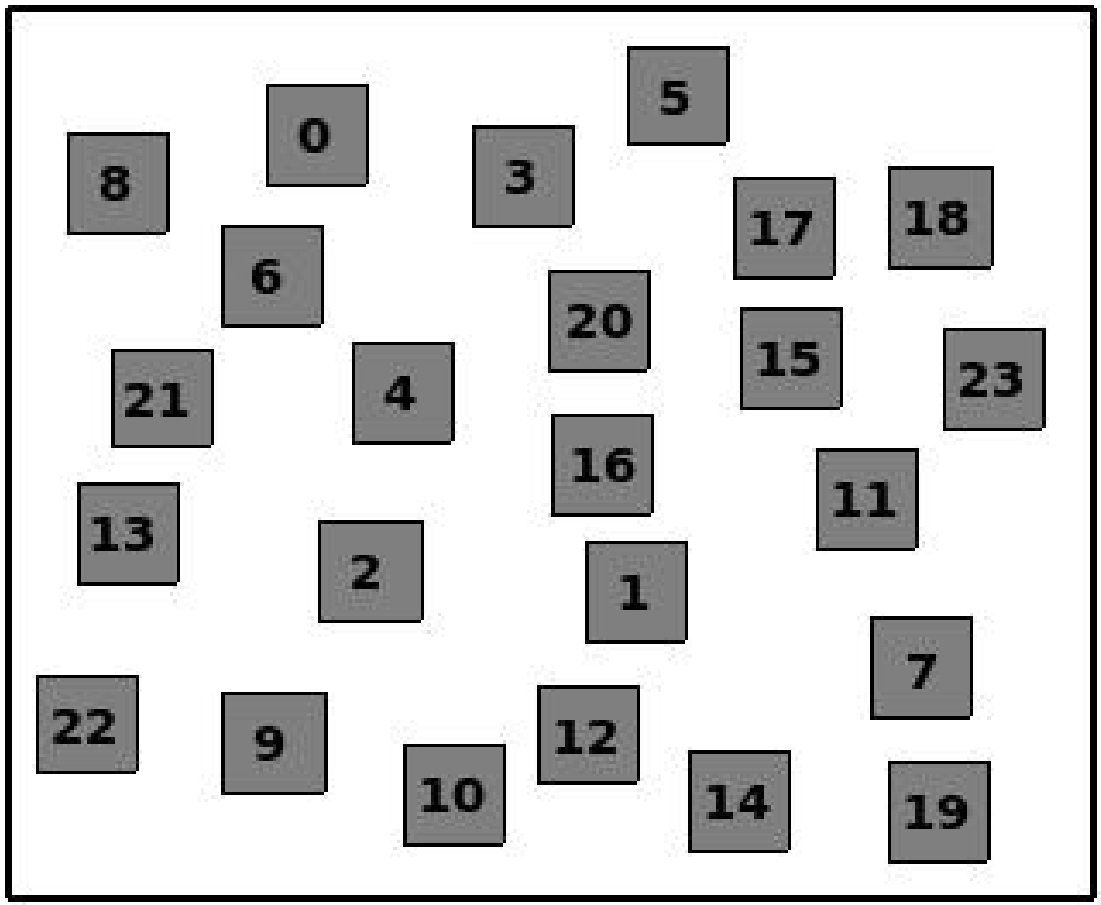
$T_1 < T_2$

Determination of Shared Caches

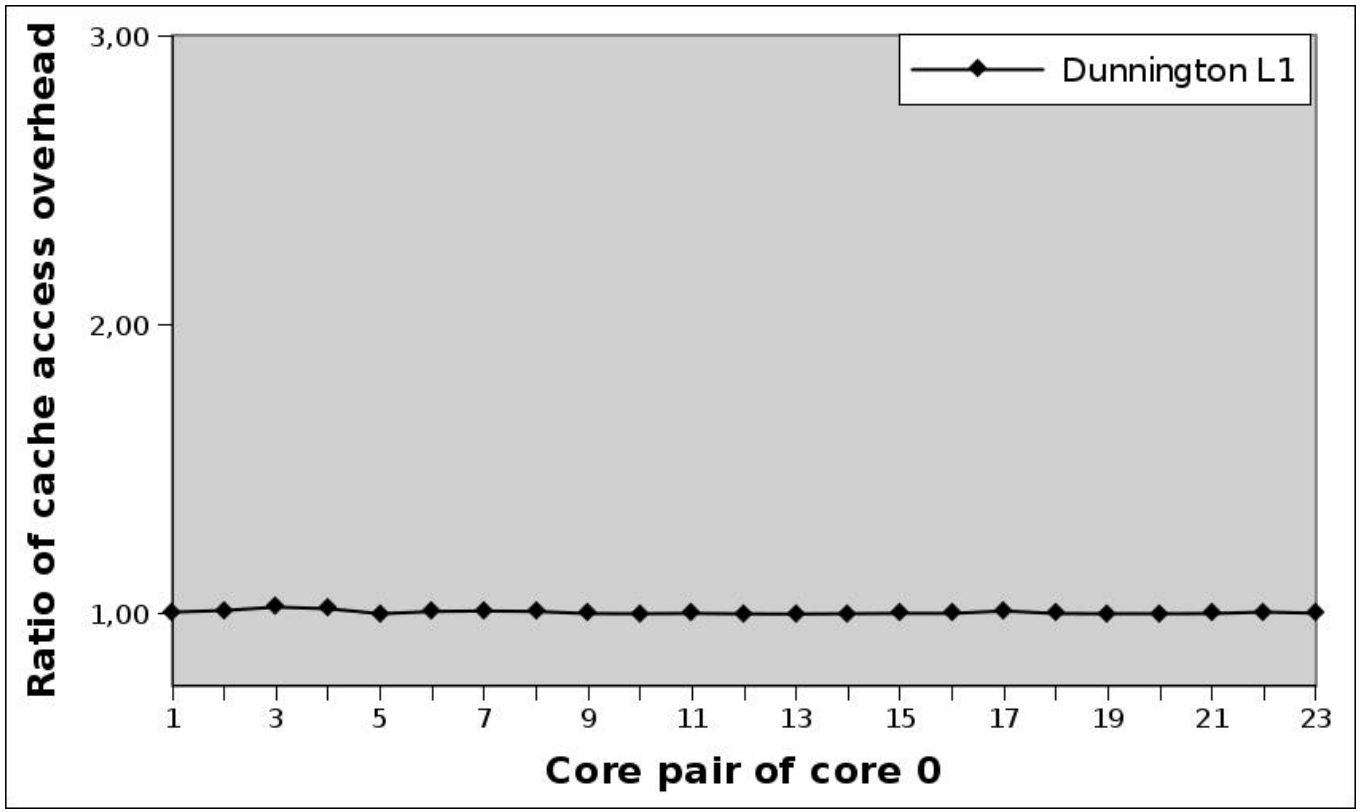
Algorithm

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

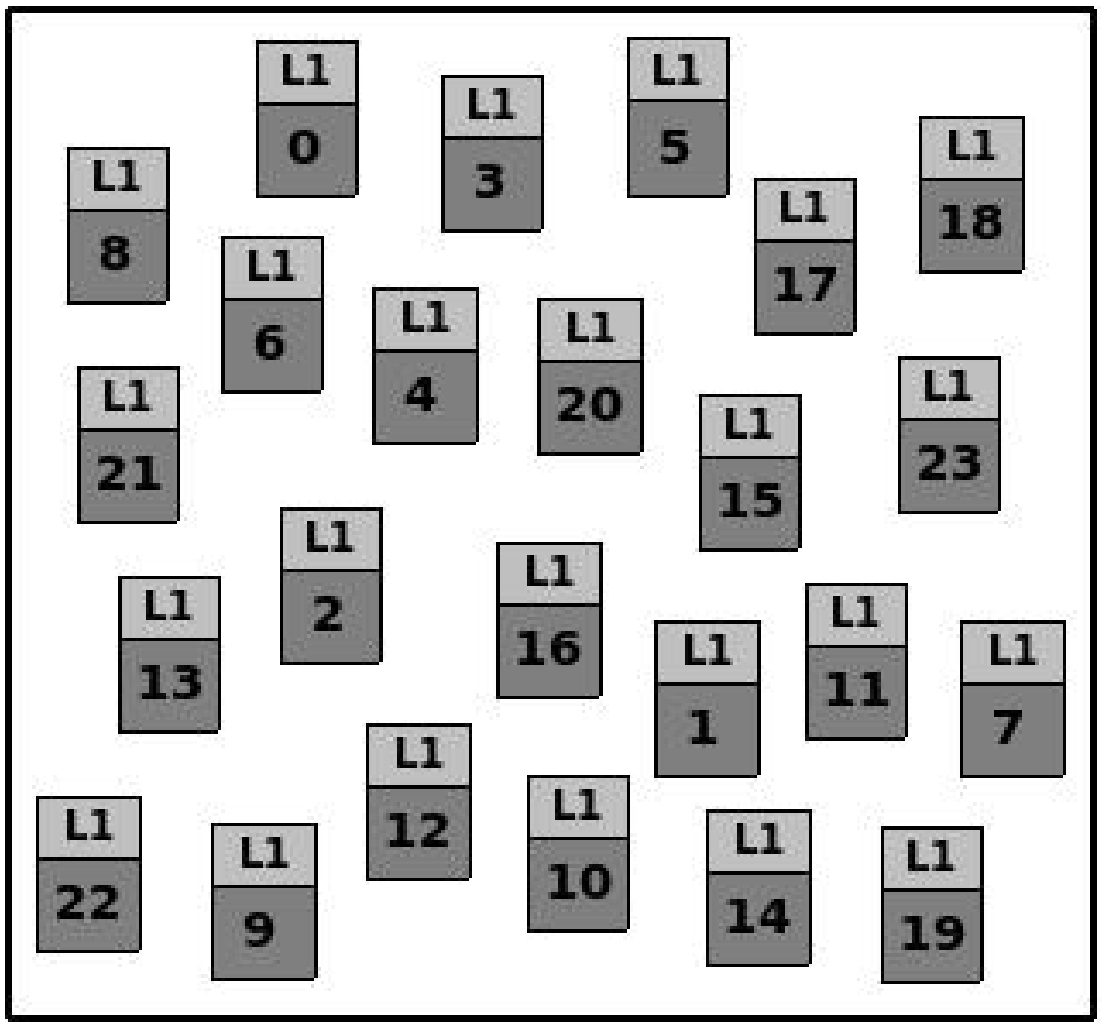


Dunnington Example: L1 Results

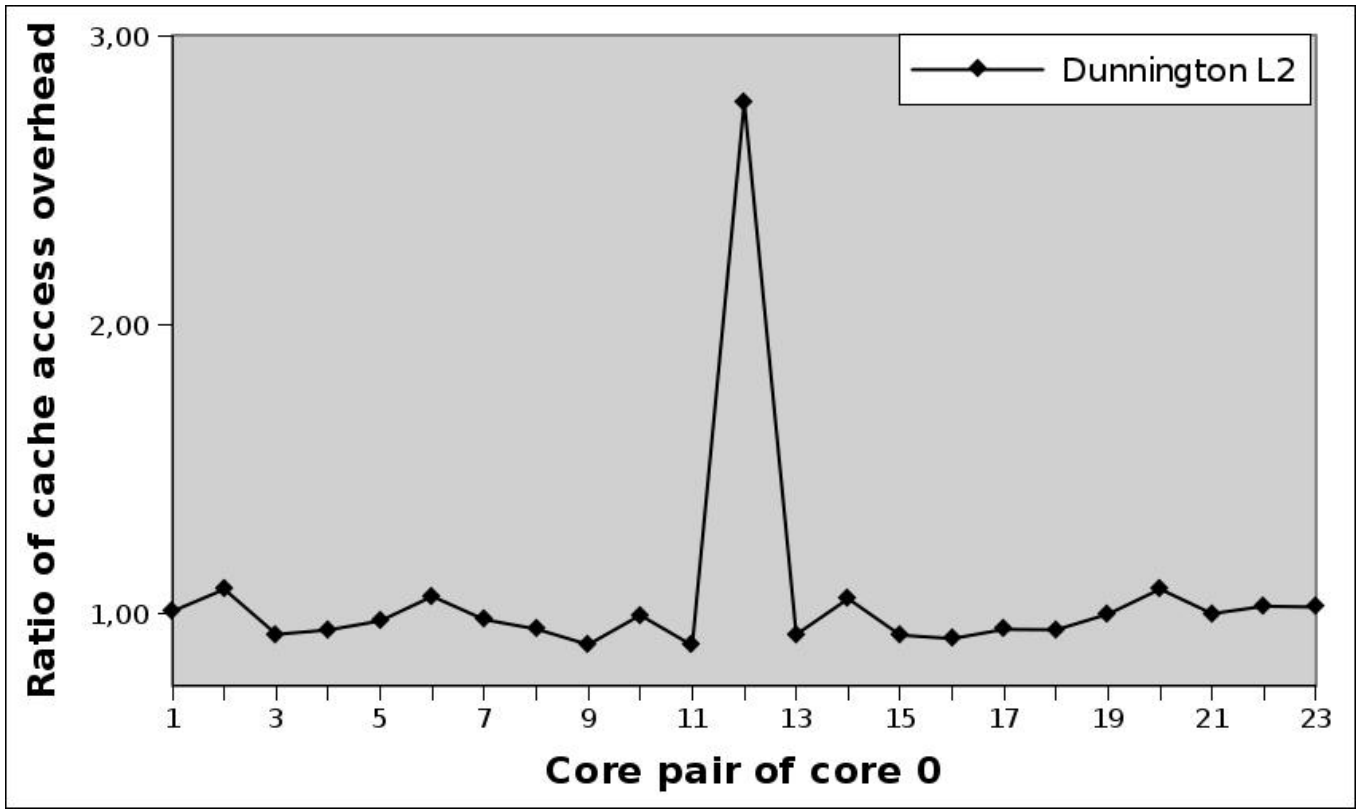


$L1 \rightarrow$ not shared

Topology with L1

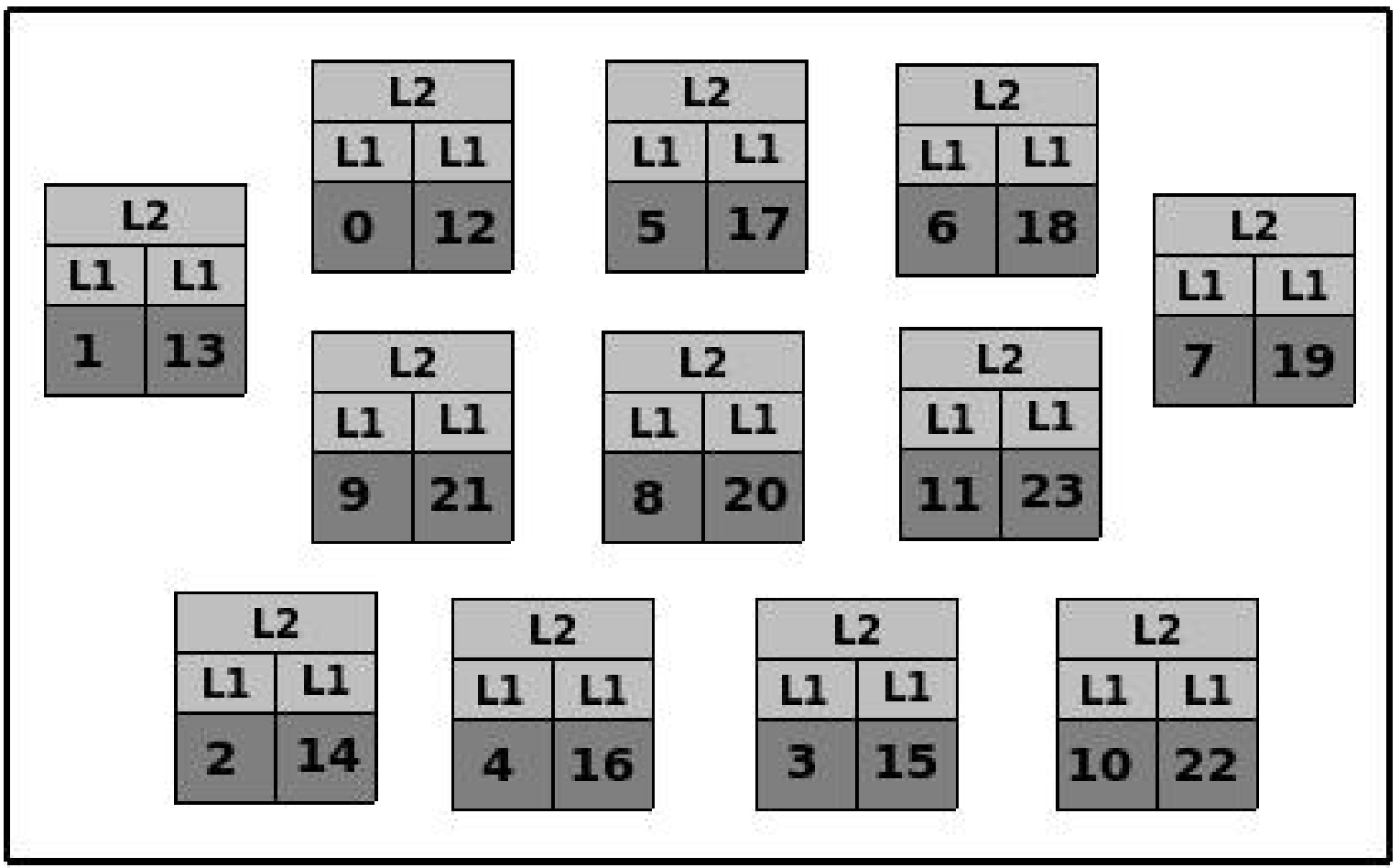


Dunnington Example: L2 Results

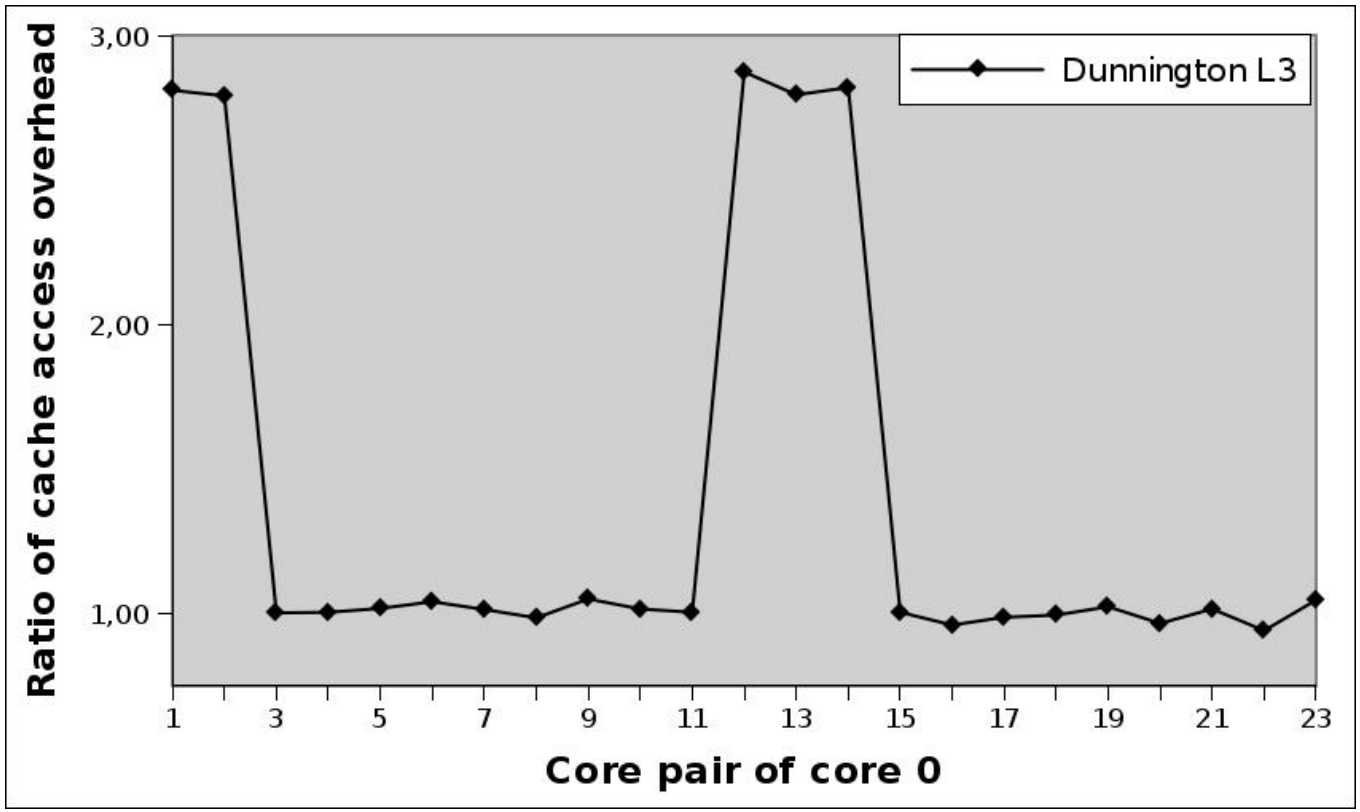


$L2 \rightarrow$ shared by 0 and 12

Topology with L2

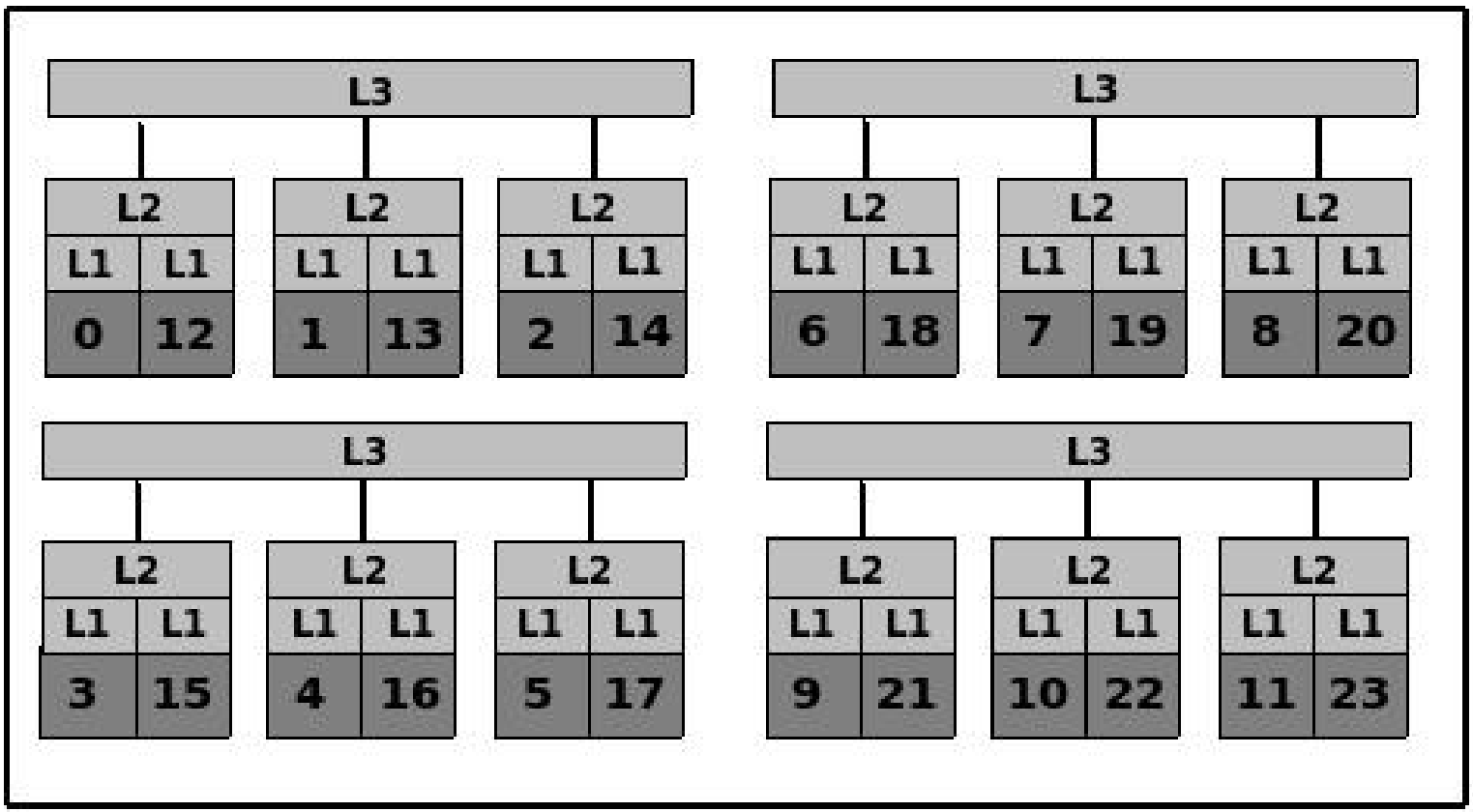


Dunnington Example: L3 Results



$L3 \rightarrow$ shared by 0,1,2,12,13 and 14

Topology with L3

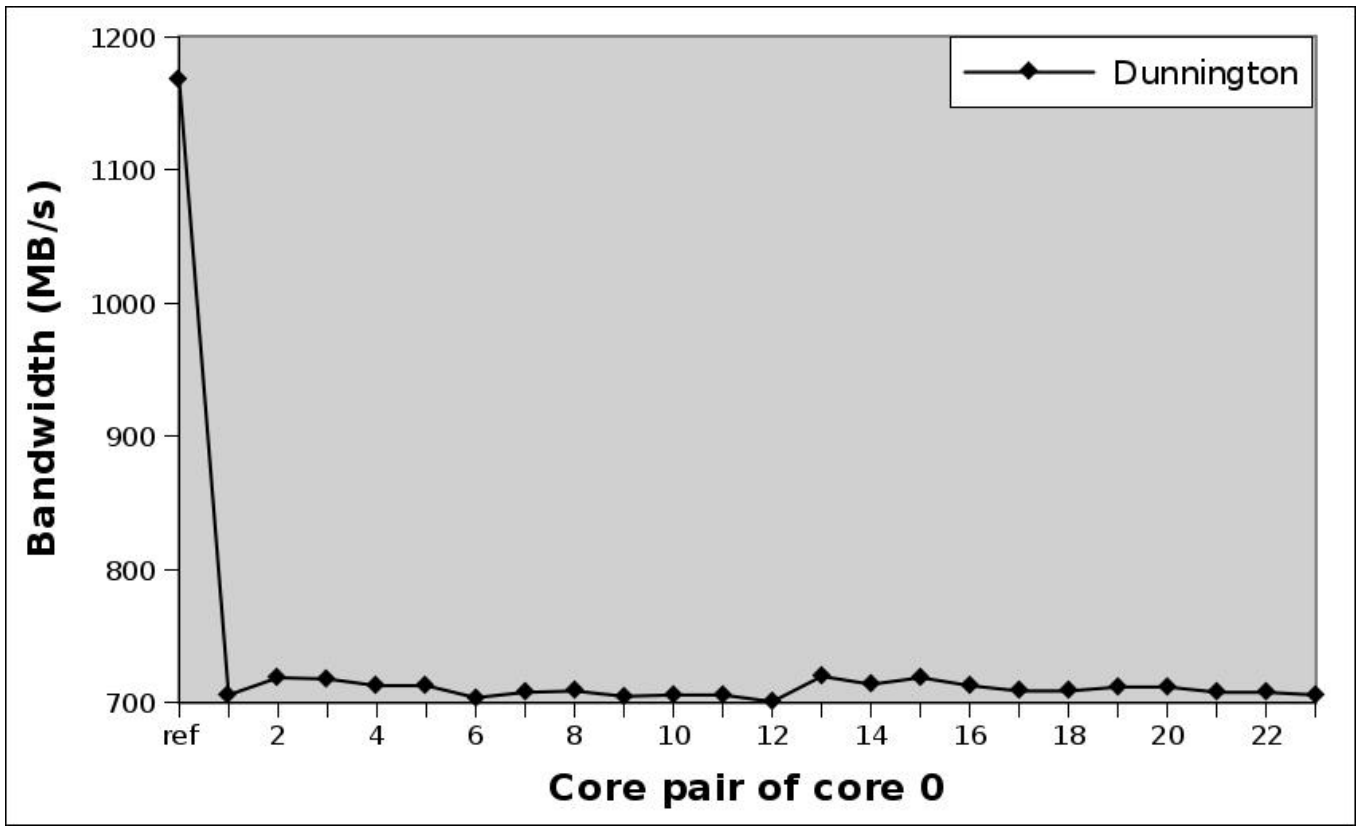


Detection of Different Memory Access Overheads

Algorithm

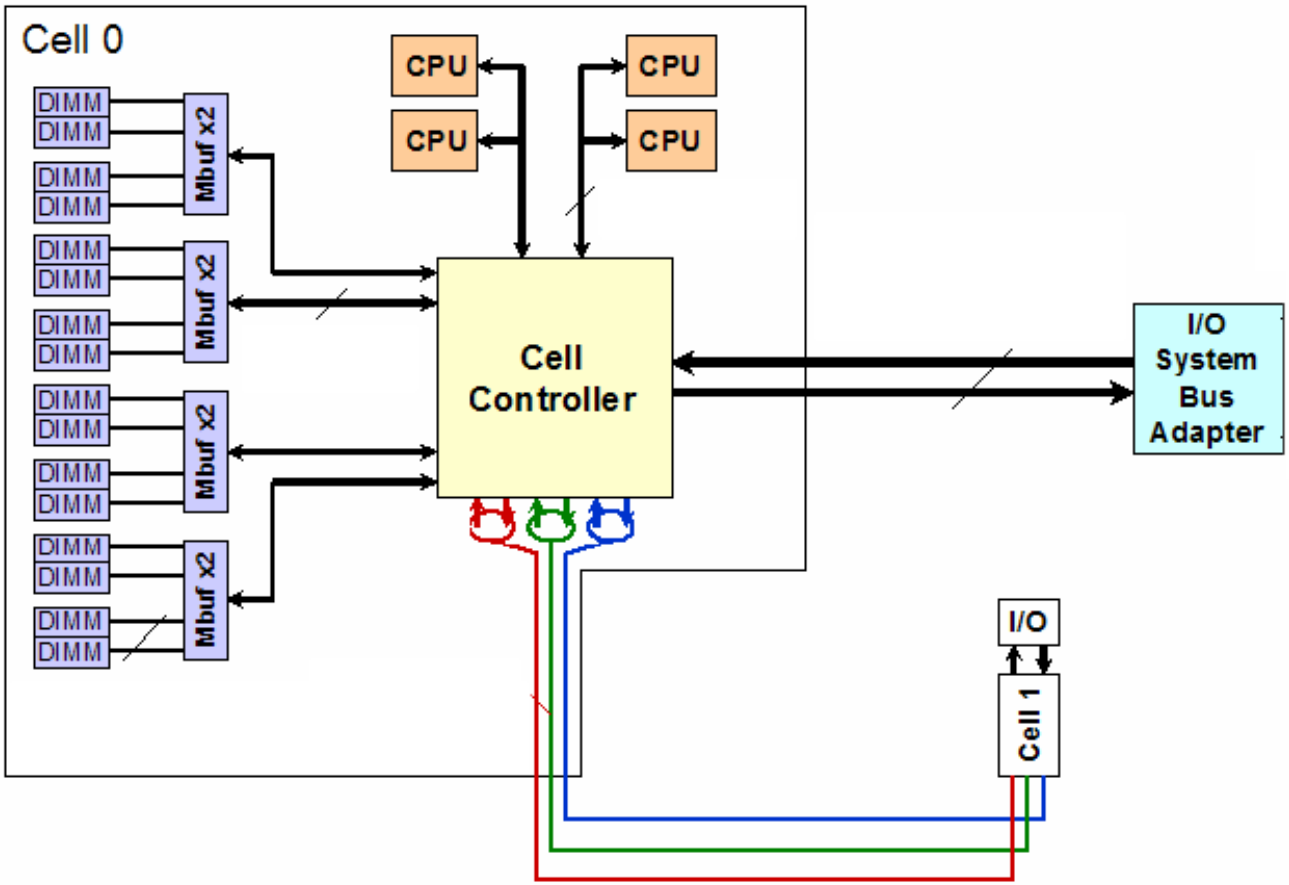
$n = 0$
ref = memory bandwidth when accessing one isolated core
foreach(pair of cores)
    b = bandwidth of one process when accessing both cores
    if($b < ref$)
        if(b similar to a given $BW[i]$)
            Add the pair to $P_m[i]$
        else
            $BW[n] = b$
            $P_m[n]$ = The used pair
            $n = n + 1$

Dunnington Results

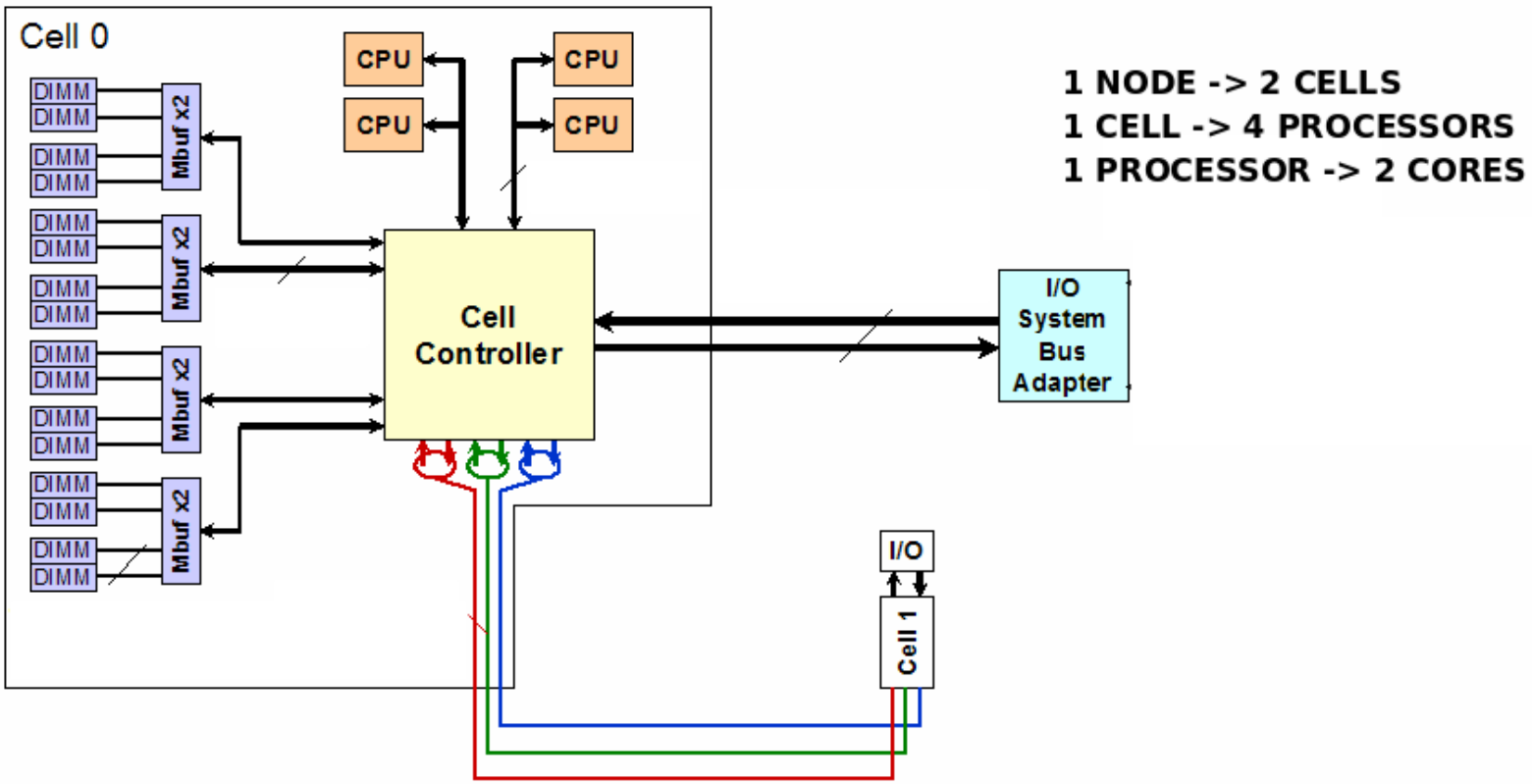


60% of bandwidth when accessing by pairs

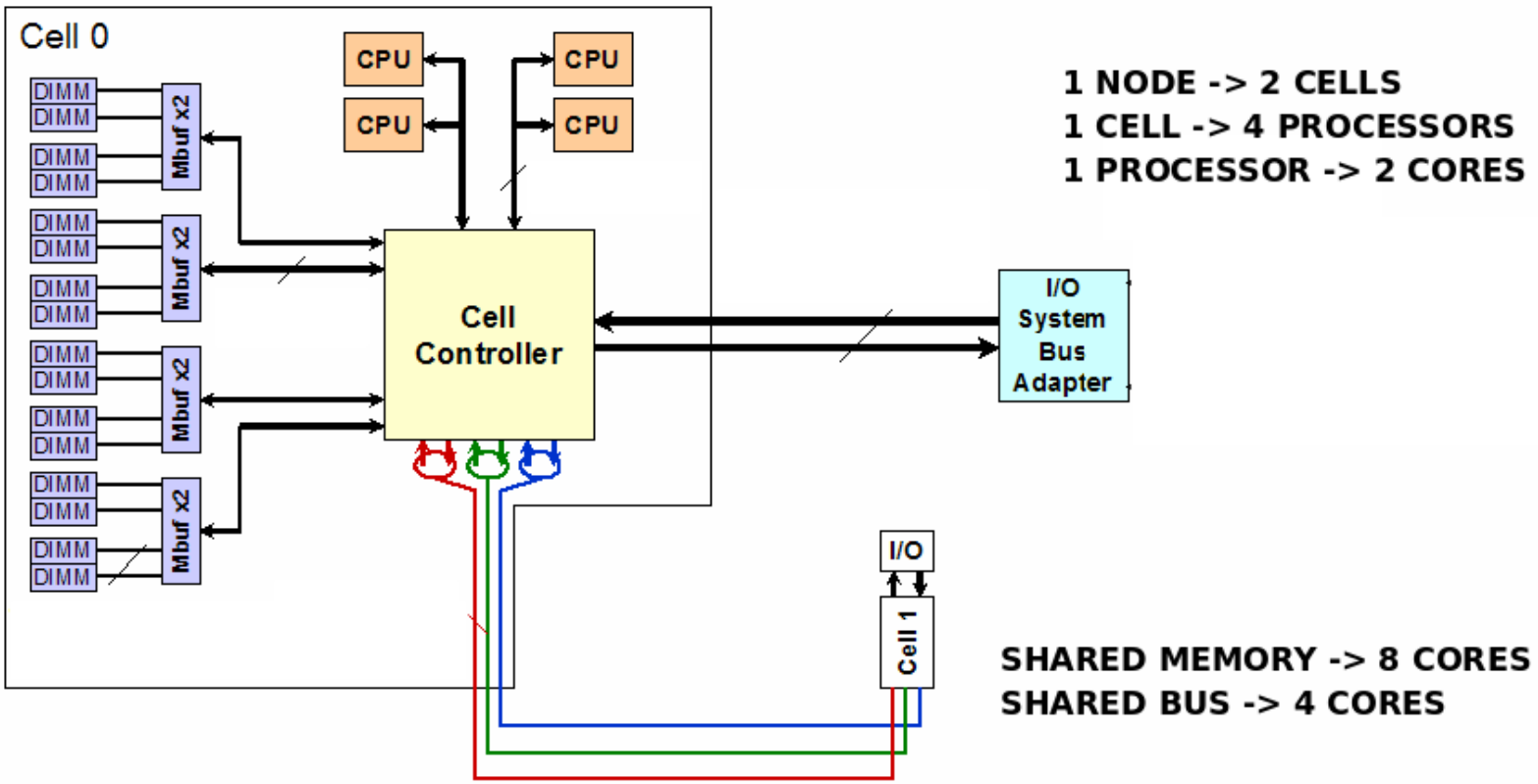
Finis Terrae Architecture (I)



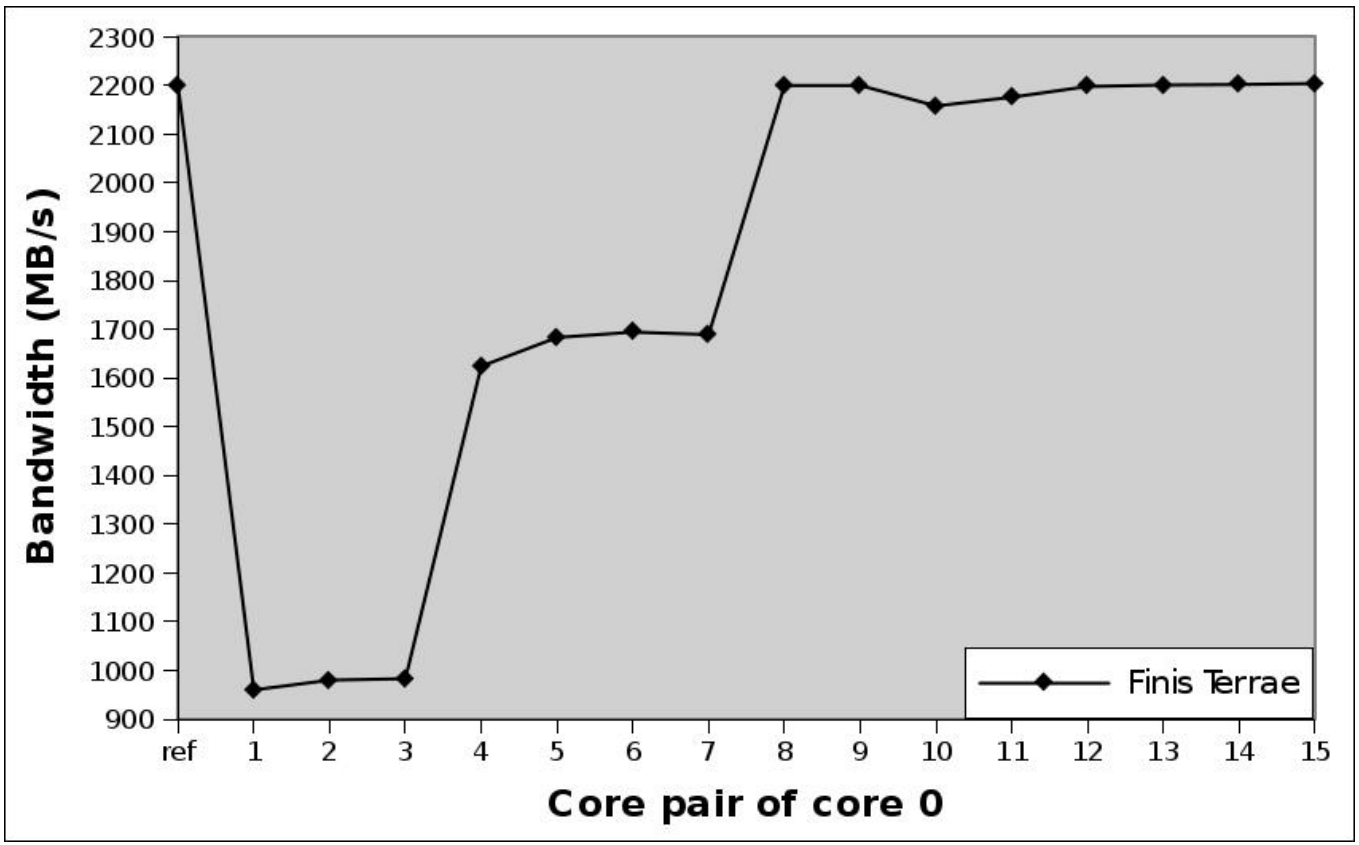
Finis Terrae Architecture (II)



Finis Terrae Architecture (and III)



Finis Terrae Results (I)

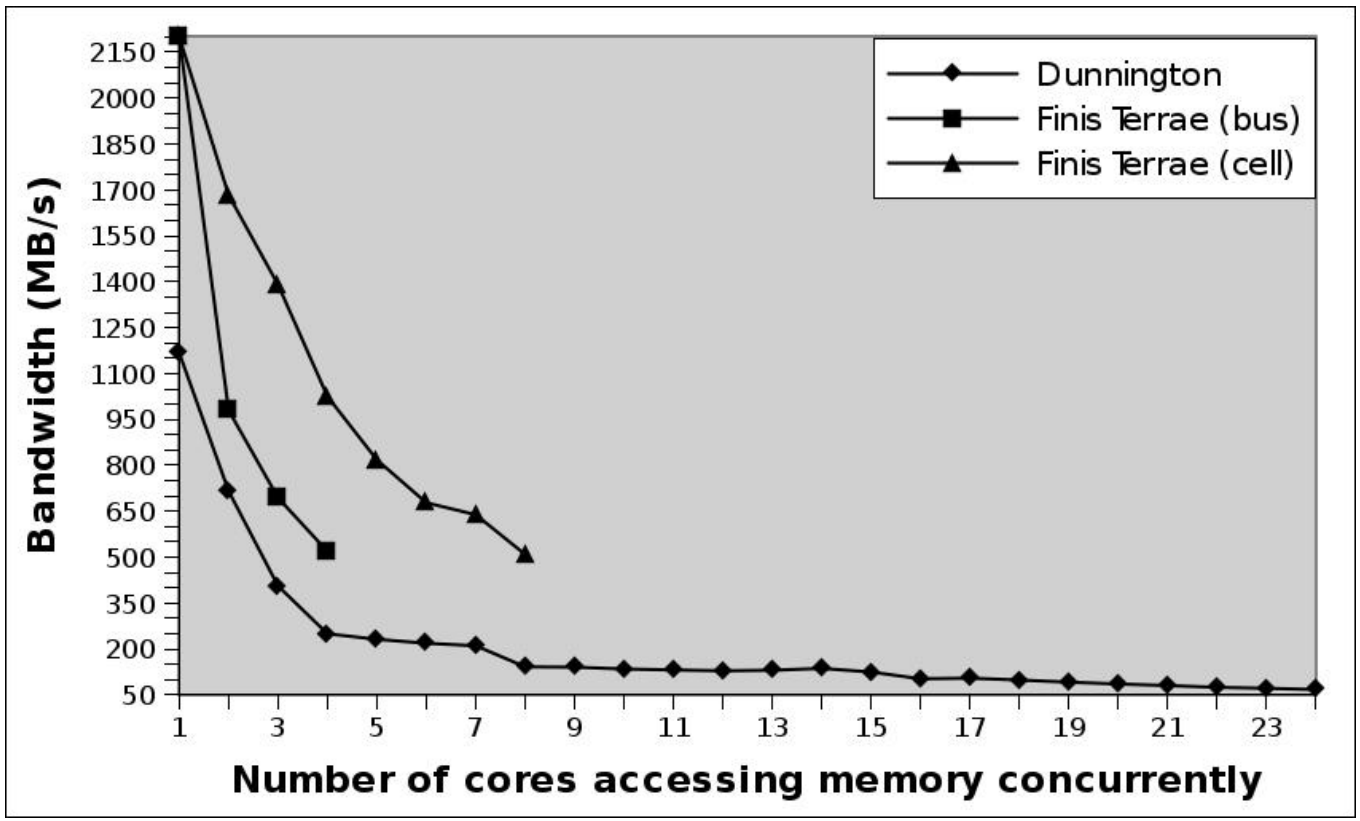


Finis Terrae Results (and II)

Memory Bandwith in Finis Terrae

- Isolated acesses
  - 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

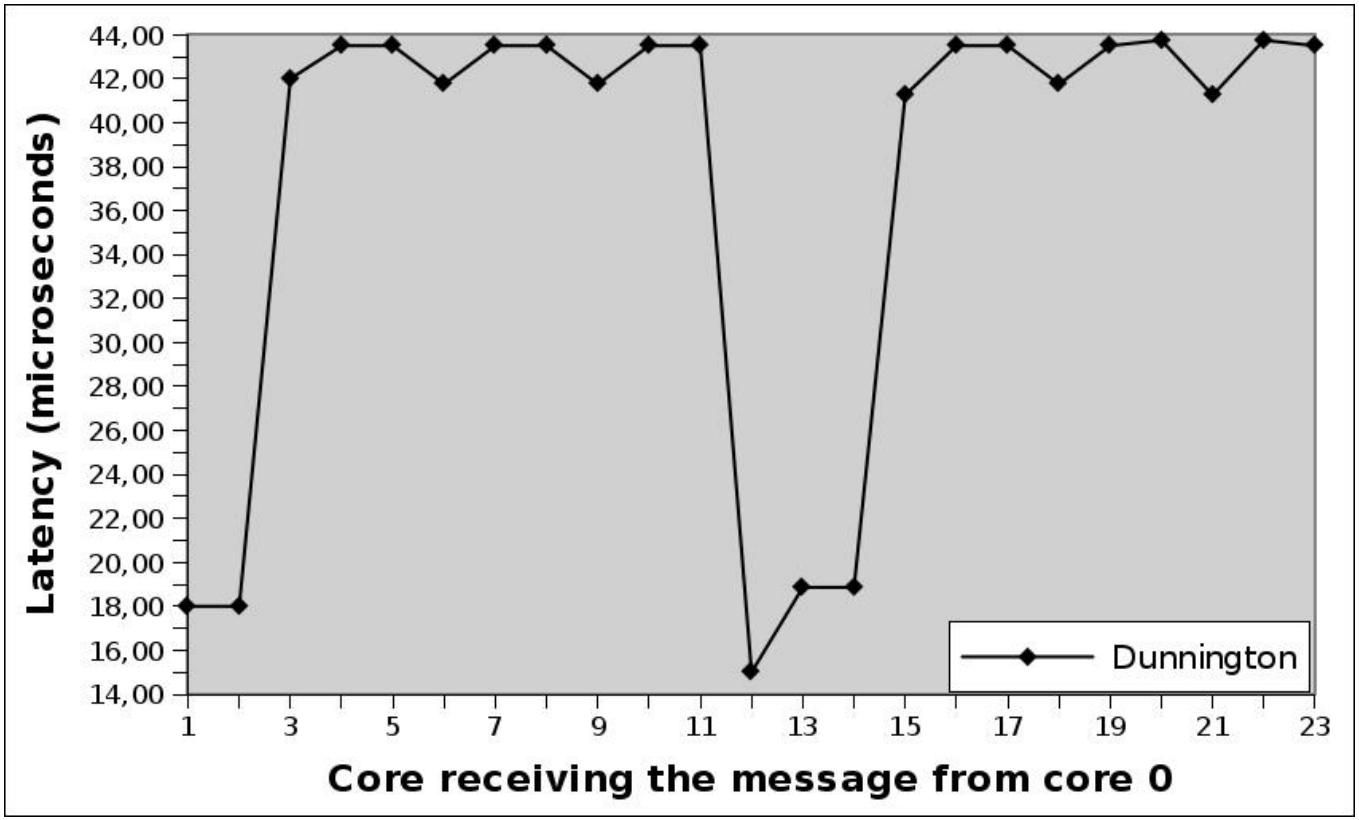


Determination of Communication Layers

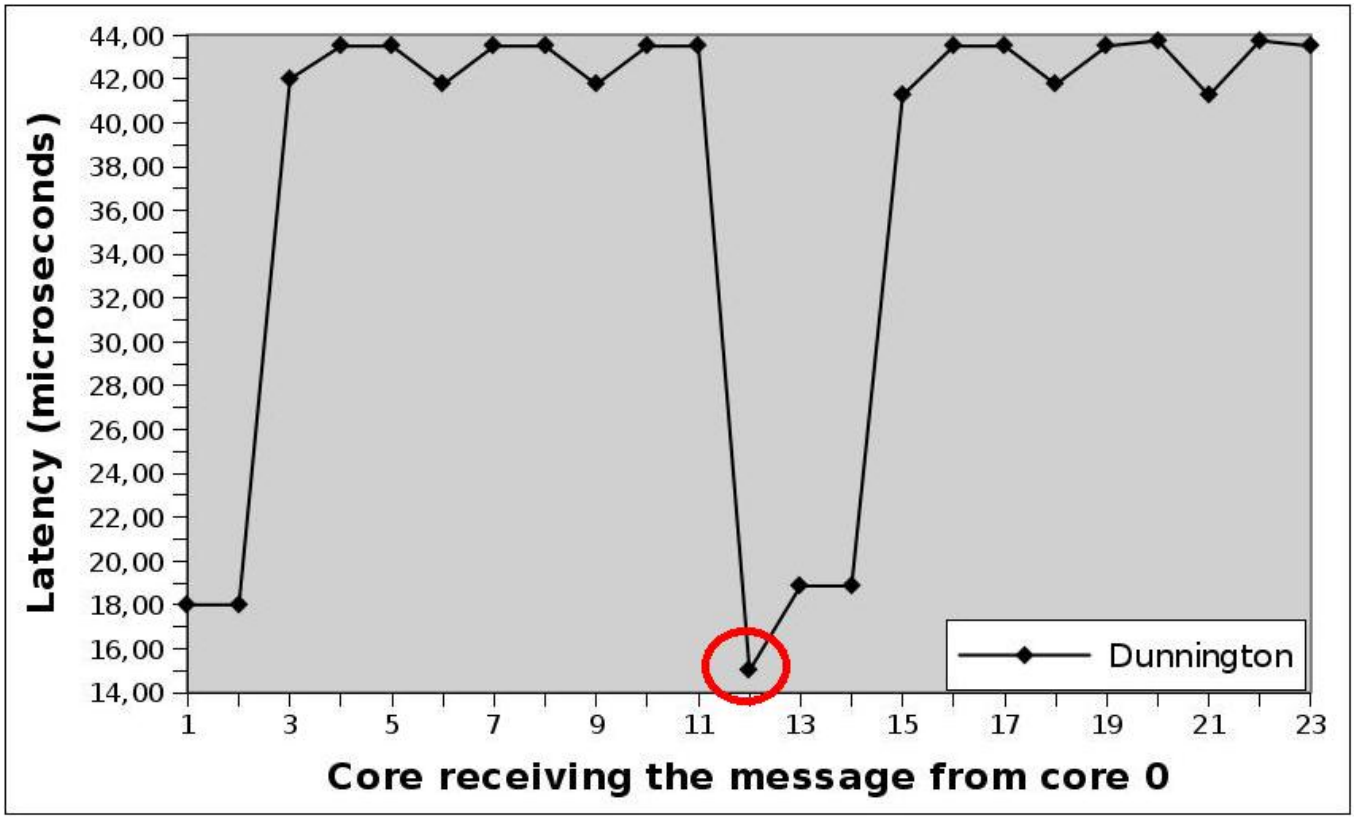
Algorithm

$n = 0$
foreach(pair of cores)
    $l =$ latency with a message of L1 size between the two cores
    if(b similar to a given $L[i]$)
        Add the pair to $P_l[i]$
    else
        $L[n] = l$
        $P_l[n] =$ The used pair
        $n = n + 1$

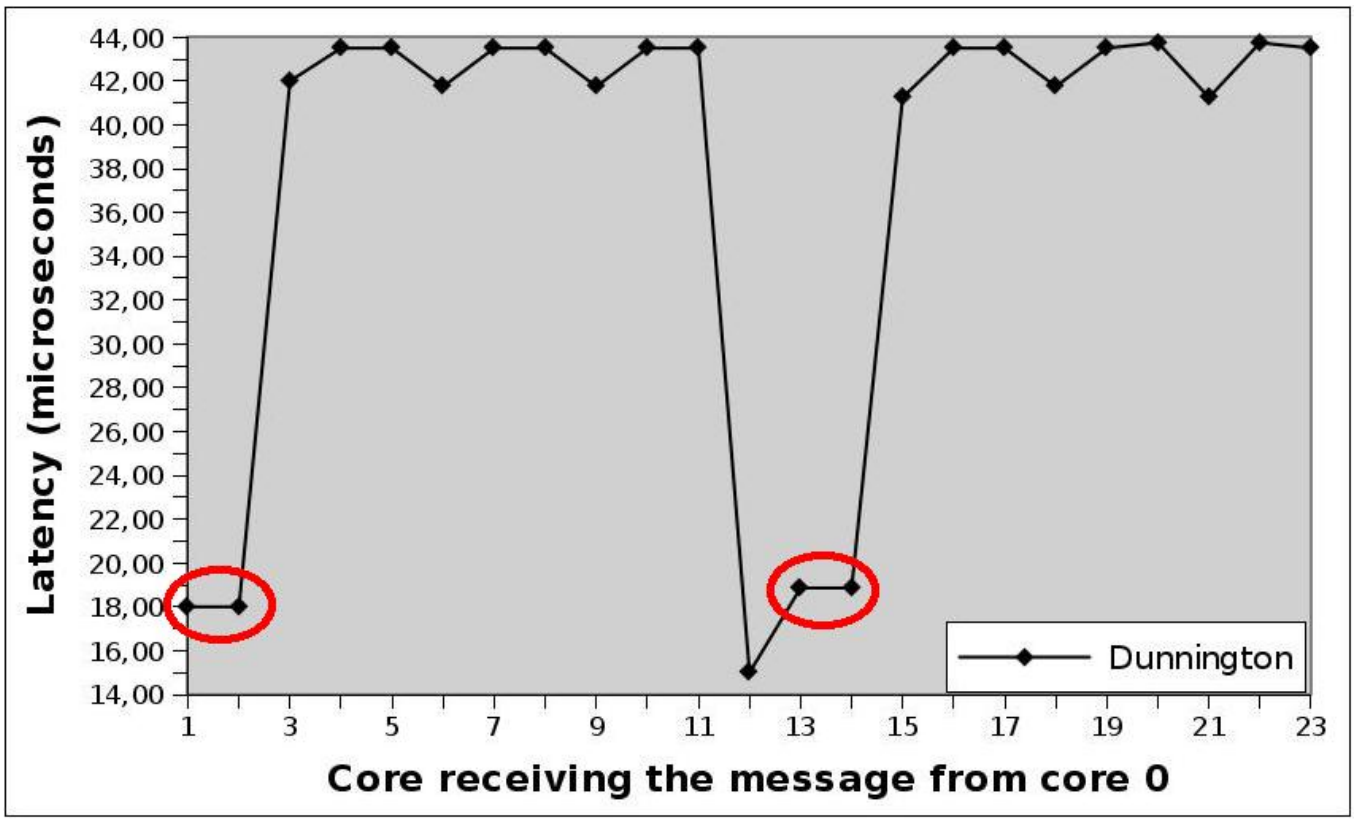
Dunnington Results (I)



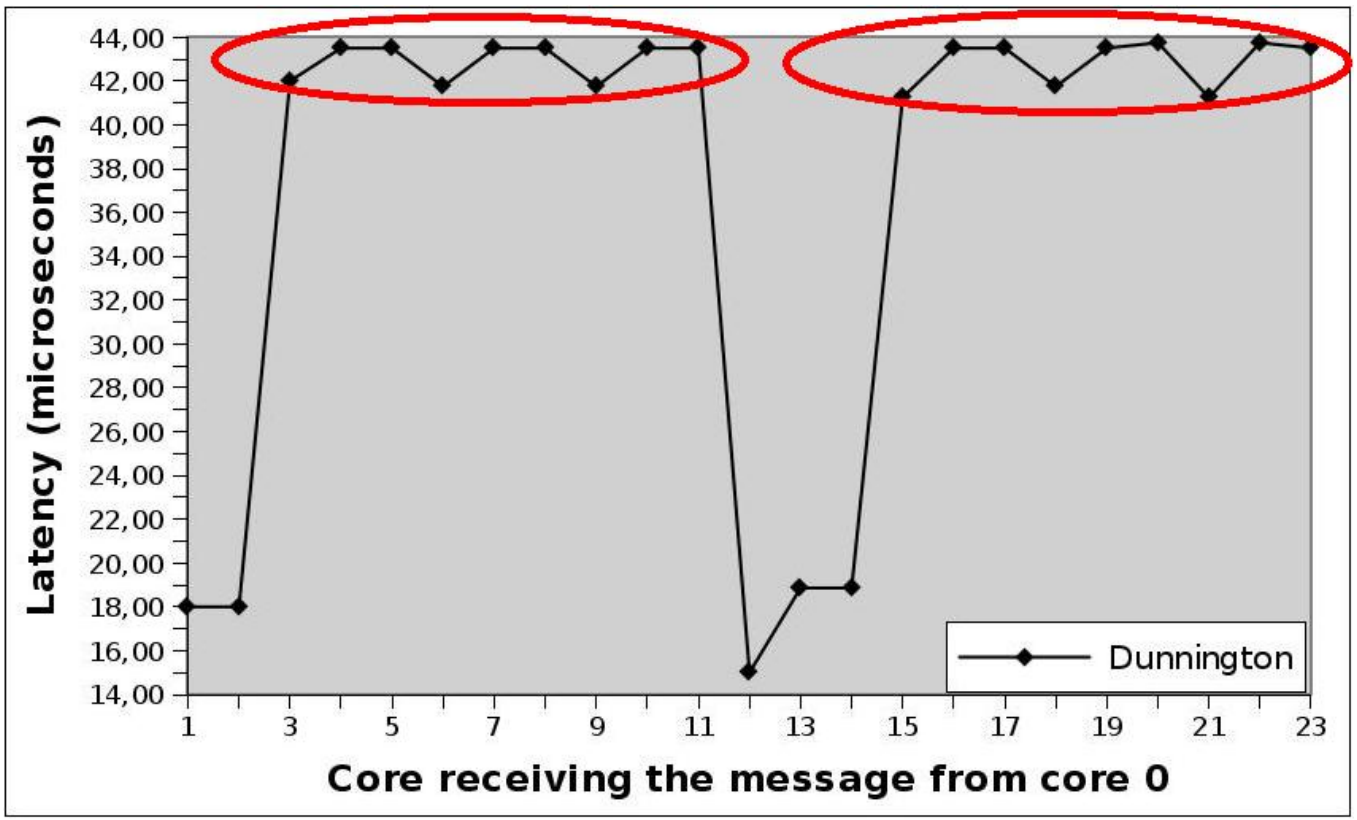
Dunnington Results (I)



Dunnington Results (I)



Dunnington Results (I)

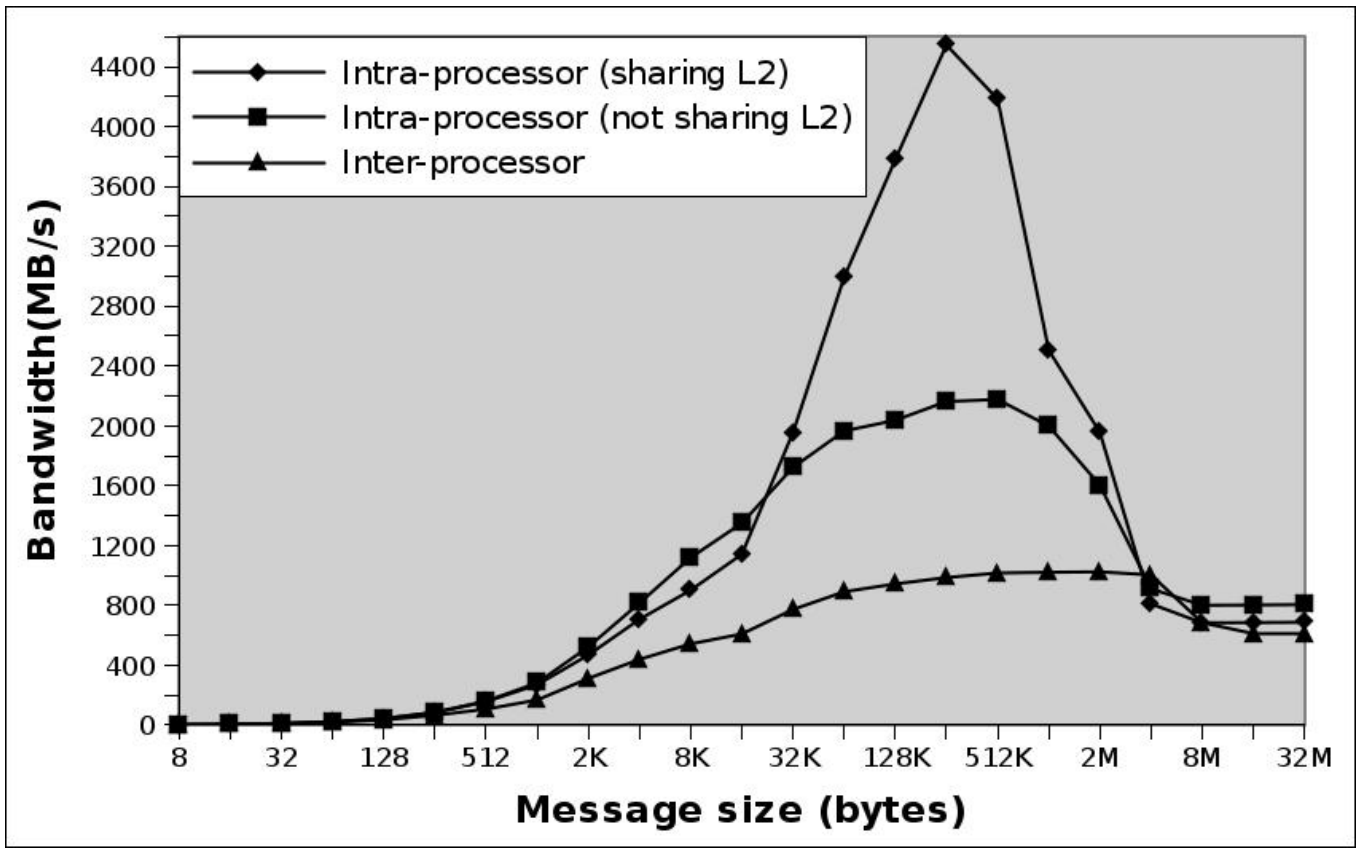


Dunnington Results (and II)

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 posible the highest bandwidth
- Inter-processor communications
  - 750 MBytes/s
  - Only 35% of posible highest bandwidth

Characterization of Communication Layers



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