Parallel computations and memory management

M. Gastineau

IMCCE - Observatoire de Paris - CNRS UMR8028
77, avenue Denfert Rochereau
75014 PARIS
FRANCE

gastineau@imcce.fr

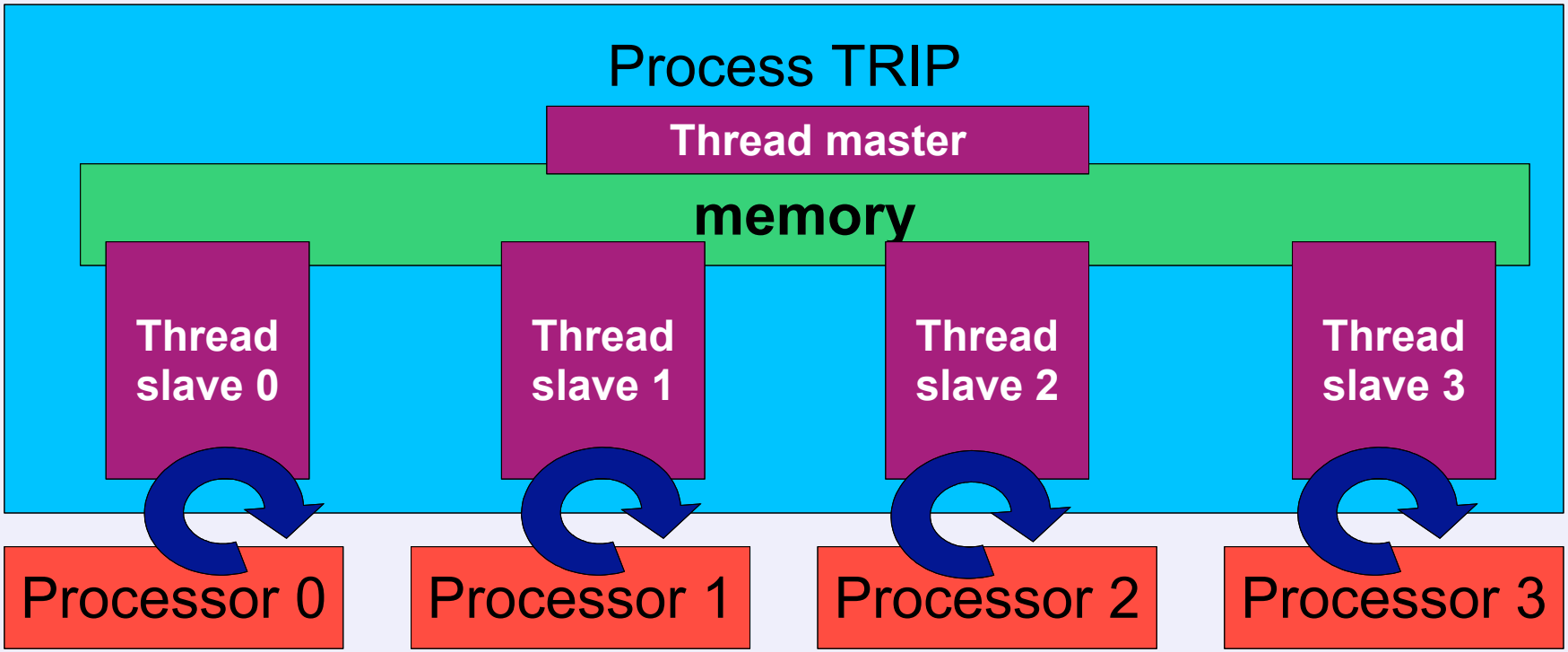
Contents

- Parallel computations
  - shared memory
  - distributed computations

- Memory management
  - garbage collection
  - reference couting
  - explicit memory management

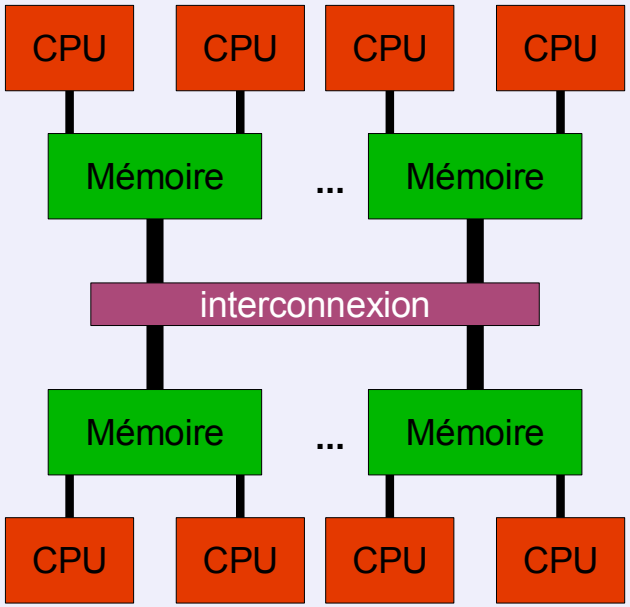
Shared memory systems

- Split the work on several processors

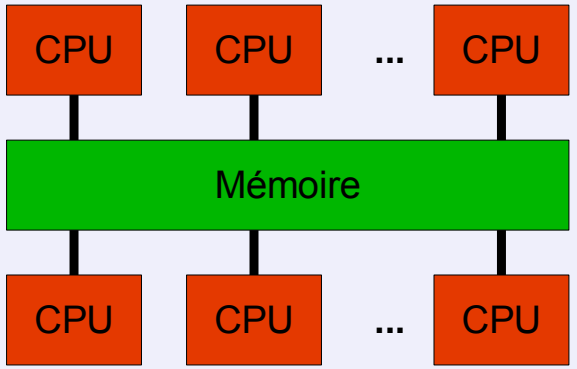


- All threads will share same memory space ➪ no copy !

- OpenMP
  - Simplest way to parallelize regular loops but performance could depend on the architecture of the computer and the location of the data

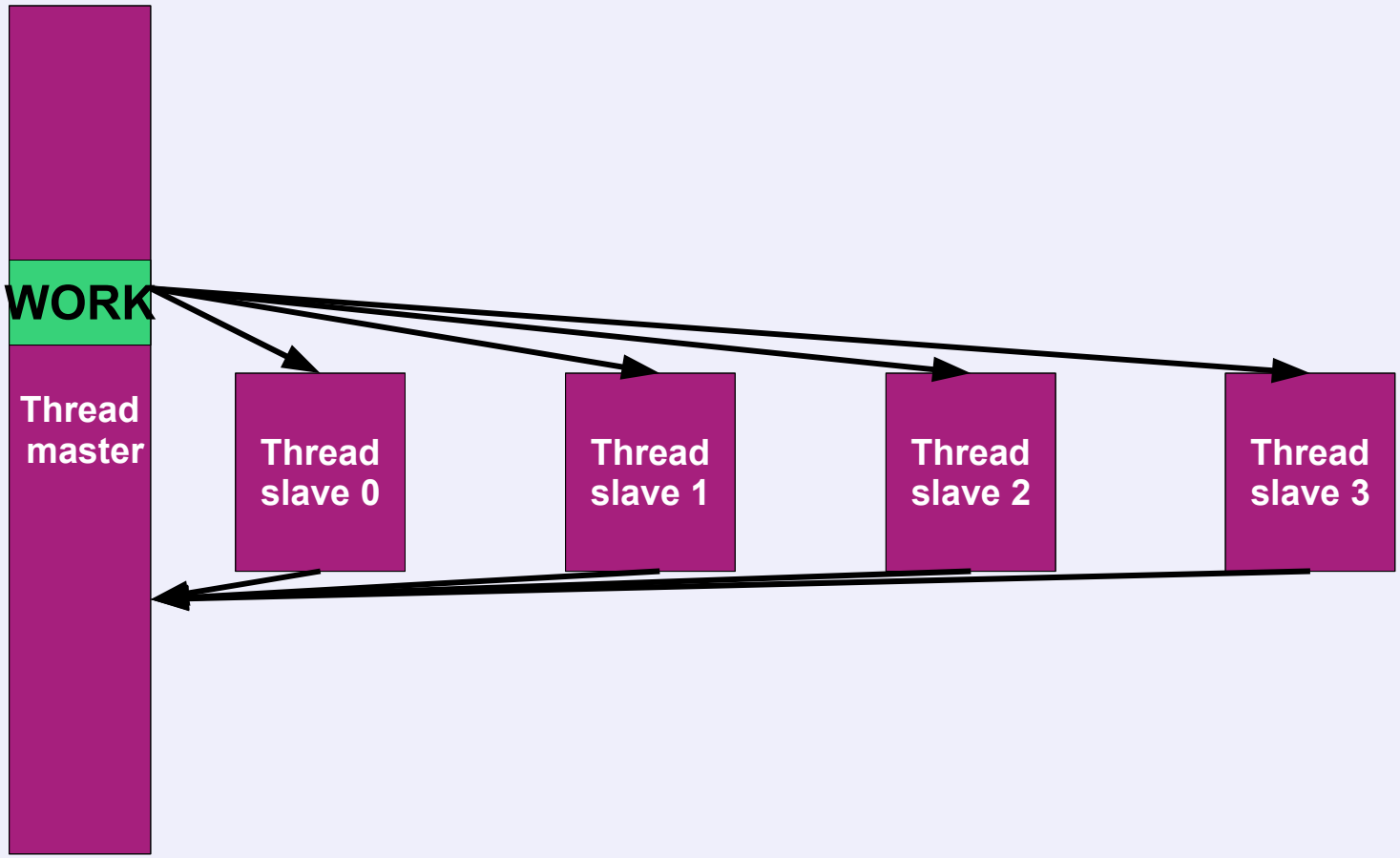


NUMA

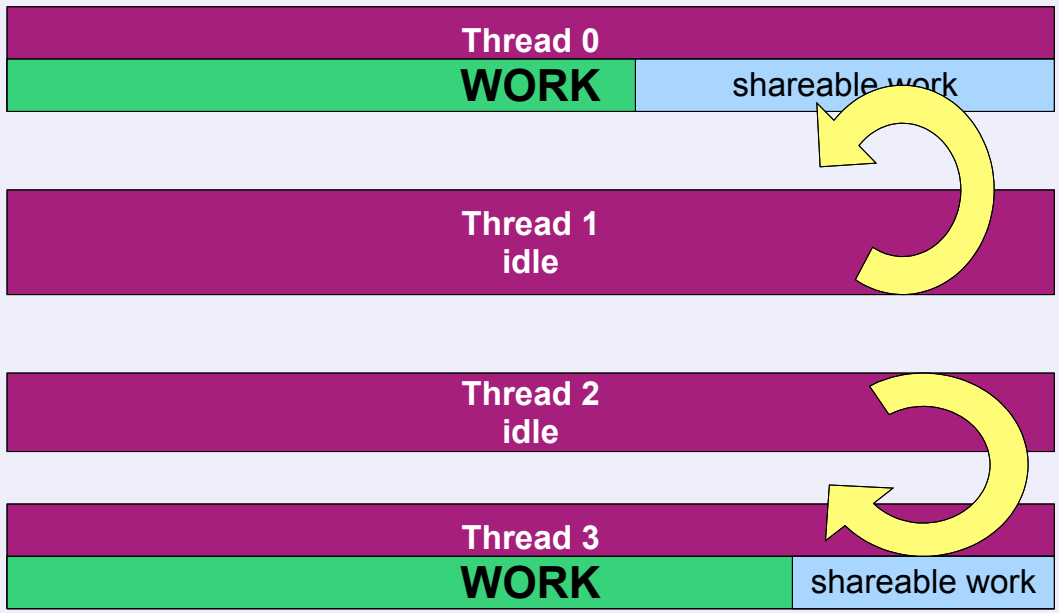


UMA

- Pthreads API
  - set of very basic functions ➪ require large additional lines of code
  - master-slave model
    - The master splits the work and gives it to each slave thread

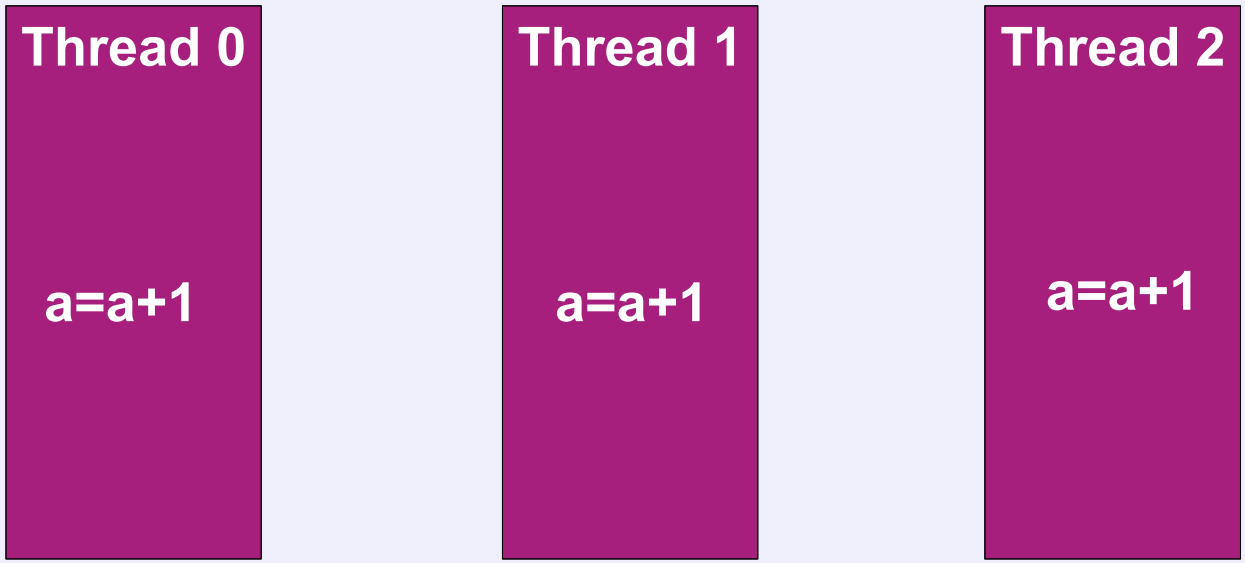


- Pthreads API
  - work stealing model
    - Each thread has a queue of available work that can be done by others.
    - Unused thread get part of the work from the queue of others threads.
    - load imbalance disappears.
    - Used in *TRIP* for irregular tasks using lock-free technique to access the queue



write to a shared vaiable

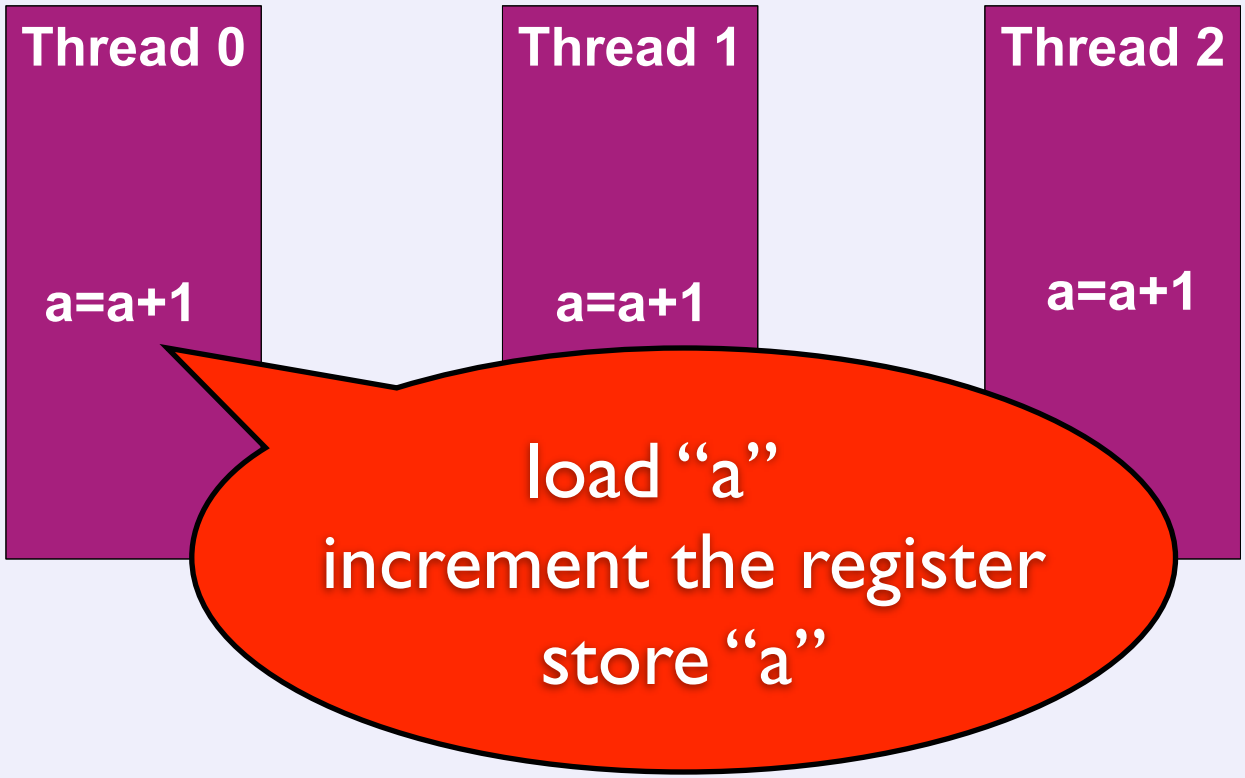
- a = 1



- value of a ?

write to a shared vaiable

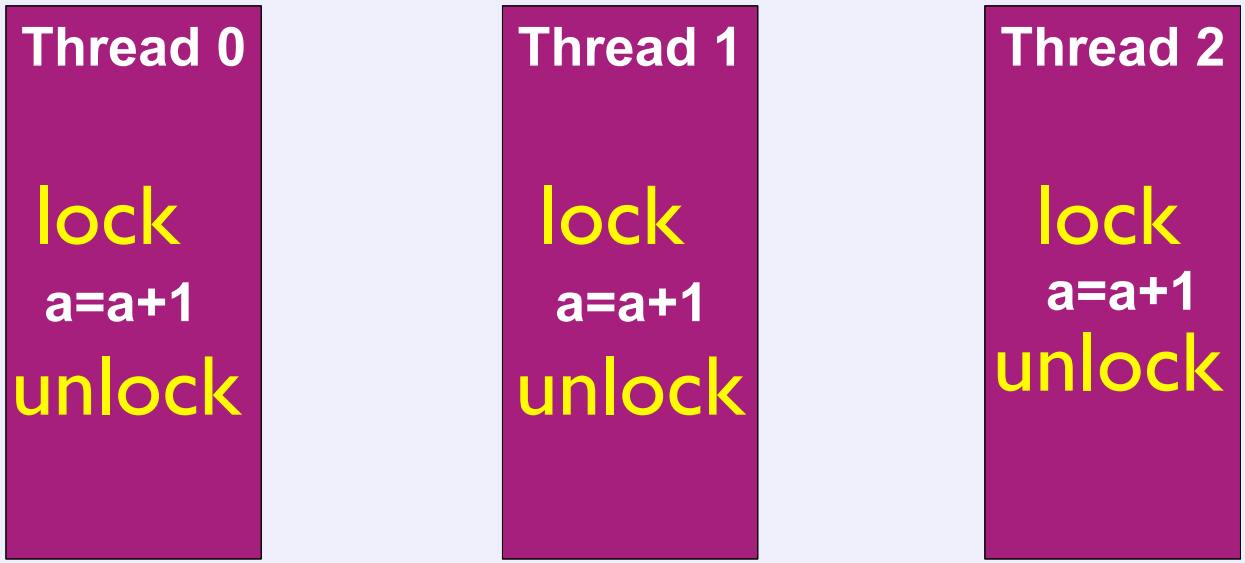
- a = 1



- value of a ?

write to a shared vaiable

- a = 1

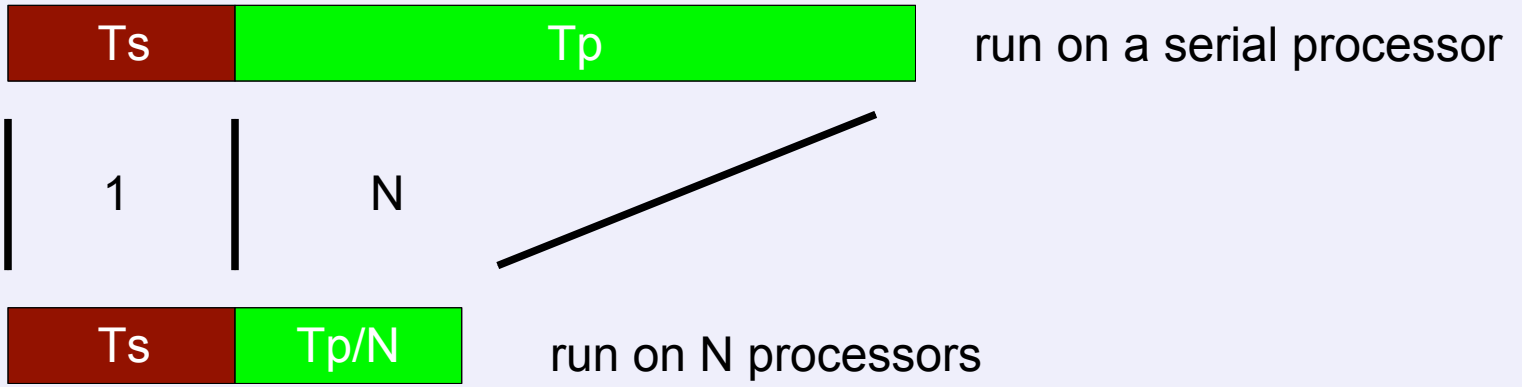


- value of a ? 4

Speedup

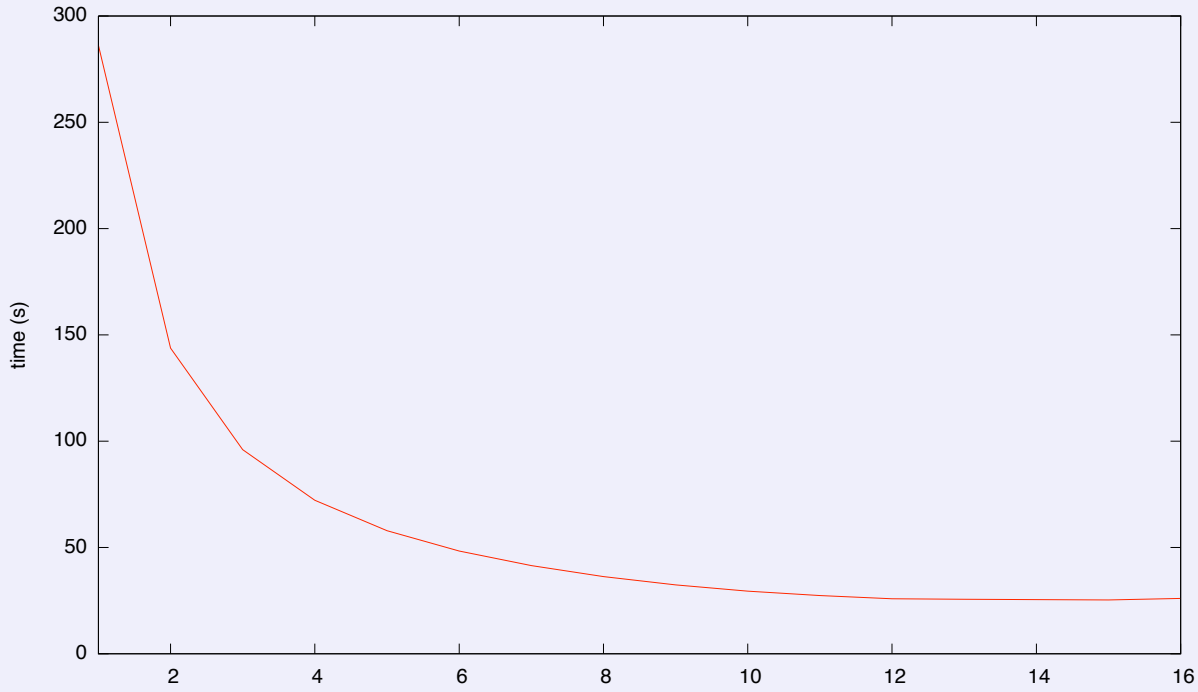
- Amdhal's law
  - Ts : time of the serial section
  - Tp : time of the parallelized section
  - N : number of elementary processor (PE)

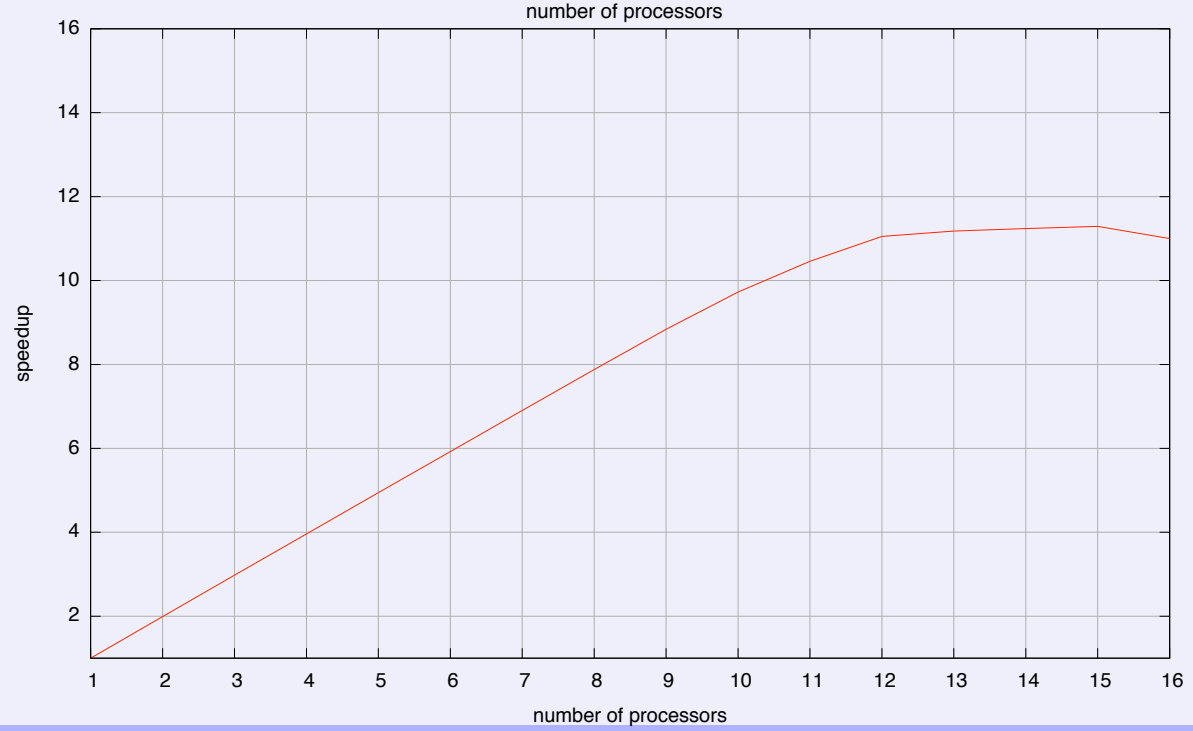
$$Speedup = \frac{1}{T_s + T_p/N}$$



- As an example, if Ts is only 10%, the problem can be sped up by only a maximum of a factor of 10, no matter how large the value of N used.

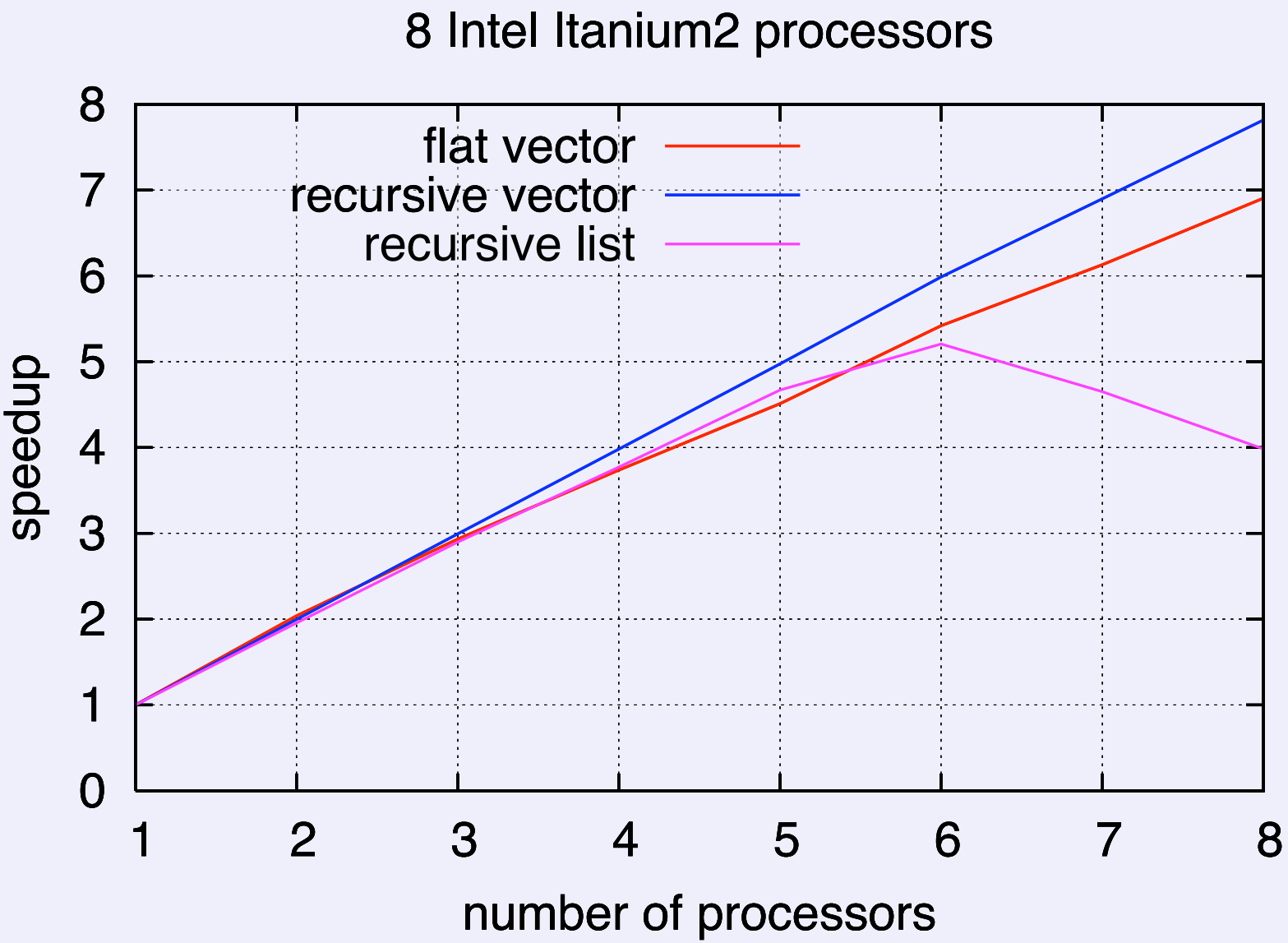
Example of speedup





parallelization of the product

V(10 variables, 57000 termes)**2



distributed computation

- Split work on several computers
  - exchange messages between computers : MPI
  - problem of the communication costs

- Example of distributed specialized algebra manipulators
  - distribute terms over nodes
    - ParFORM
    - CABAL
  - disribute task over the grid
    - GAP

Communications with general computer algebra system

- specialized system could not solve all problems !
  - need to exchange data between general and specialized systems
- MathML protocol
  - easy to produce
  - very verbose but supported by all general systems

$1 + 3x + 4x^2$ $\longleftrightarrow$

```
<math xmlns='http://www.w3.org/1998/Math/MathML'>
  <apply id='id10'>
    <plus/>
    <cn id='id1' type='integer'>1</cn>
    <apply id='id4'>
      <times/>
      <cn id='id2' type='integer'>3</cn>
      <ci id='id3'>x</ci>
    </apply>
    <apply id='id9'>
      <times/>
      <cn id='id5' type='integer'>4</cn>
      <apply id='id8'>
        <power/>
        <ci id='id6'>x</ci>
        <cn id='id7' type='integer'>2</cn>
      </apply>
    </apply>
  </apply>
</math>
```

Communications with general computer algebra system

- OpenMath protocol
  - precise semantic of each object and each function
  - very flexible protocol but emerging standard
  - will have specialized dictionaries for polynomials

$1 + 3x + 4x^2$ $\longleftrightarrow$

```
<OMOBJ>
  <OMA>
    <OMS cd = "arith1" name="plus"/>
    <OMI>1</OMI>
    <OMA>
      <OMS cd = "arith1" name="times"/>
      <OMI>3</OMI>
      <OMV name="x"/>
    </OMA>
    <OMA>
      <OMS cd = "arith1" name="times"/>
      <OMI>4</OMI>
      <OMA>
        <OMS cd="arith1" name="power"/>
        <OMV name="x"/>
        <OMI>2</OMI>
      </OMA>
    </OMA>
  </OMA>
</OMOBJ>
```

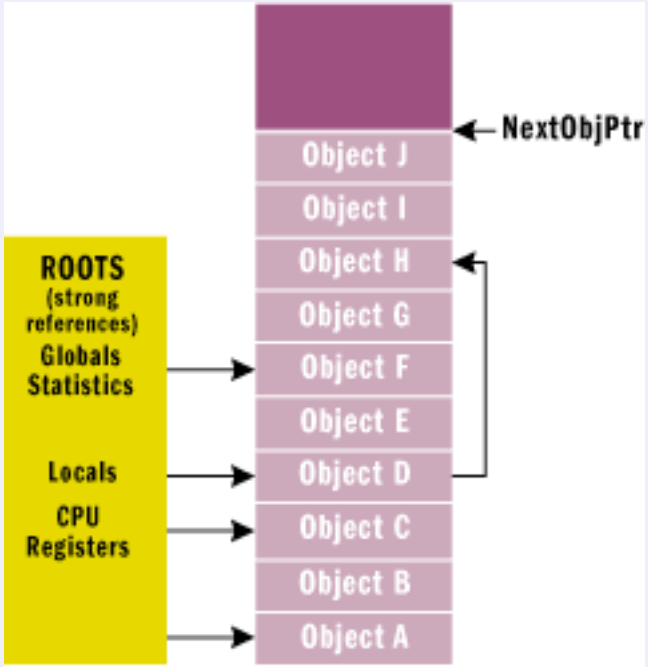
Allocated objects

- Small objects
  - element of list for the recursive list
  - element of the leaf node in the burst tries
  - arrays for small degree
- Large objects
  - arrays of coefficients for homogeneous block and flat vector
  - arrays of exponents for flat vector
- Objectives of the memory manager
  - reduce memory consumption
  - good performance

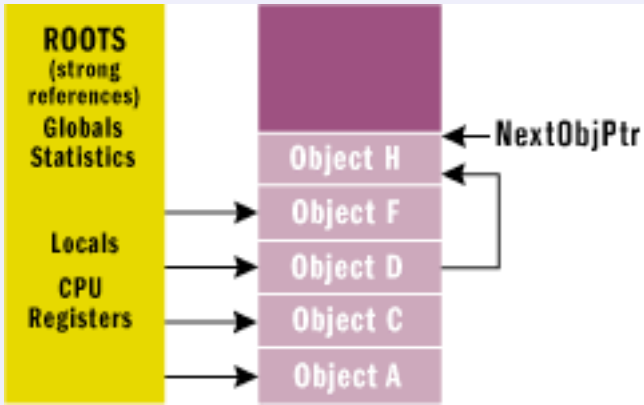
Garbage collection

- detect unused objects to recycle them
- avoid memory leaks and “double free”
- overhead
  - locate reachable objects
  - higher space consumption
  - less locality of the data
- frequency of the garbage collecting ?

before collecting



after collecting



Reference counting

- Reference counting
  - each object has a count of the number of references on it
  - the counter is incremented when a reference to it is created
  - the counter is decremented when a reference to it is destroyed
  - the object is destroyed when the counter reaches 0.

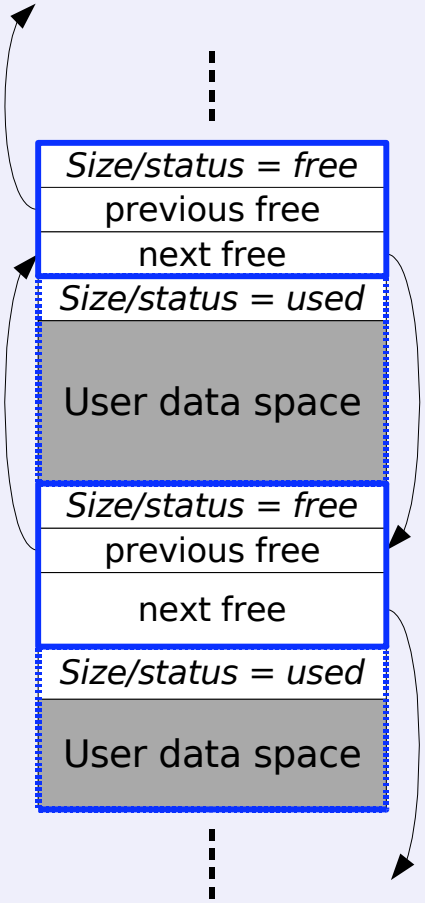
- Overhead
  - one integer for every object
  - less locality of the objects
  - need a lock in a multithread context

Explicit memory management

- system call : malloc and free
- developer should take care to every deallocations and references on objects
  - memory leak
  - “double free” bug
  - invalid reference to destroyed objects
- memory could be reused immediately
- most explicit memory managers introduce overheads
  - space
  - execution time

space overhead

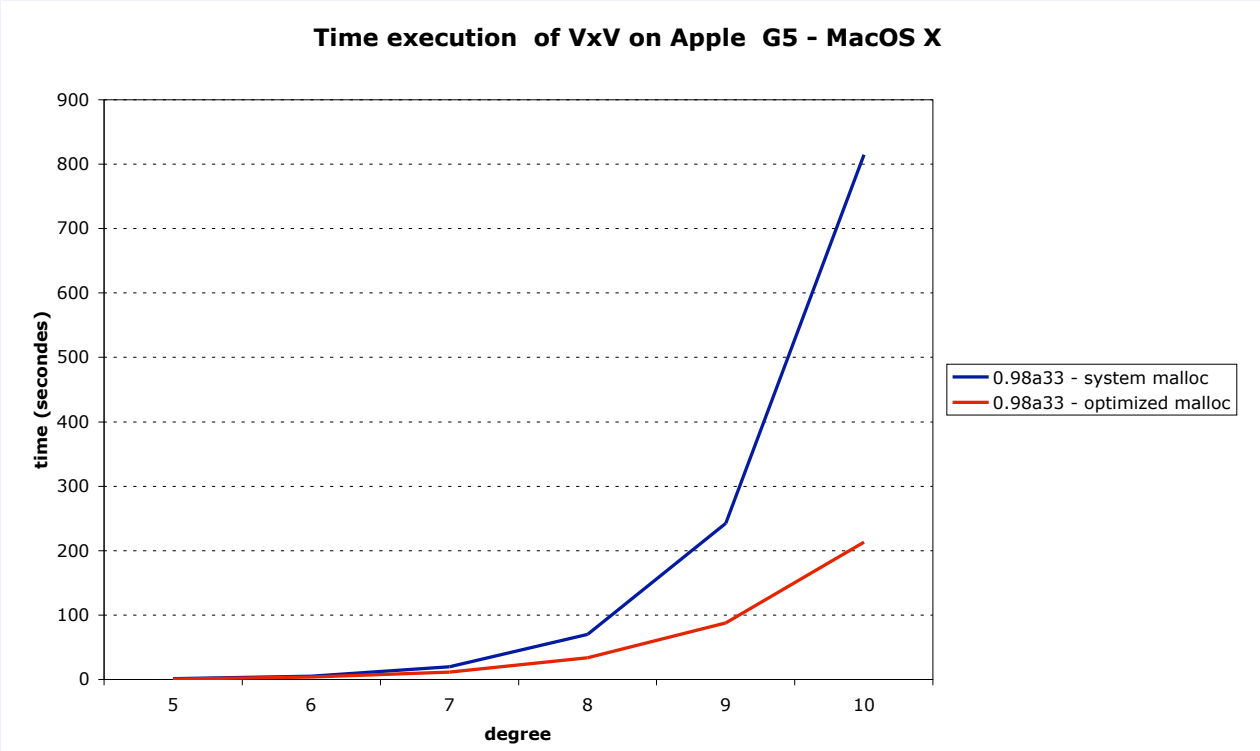
- overhead to each allocated block of memory
  - size and status
  - alignment



- e.g., Doug Lea's Malloc used in the GNU C library
  - align on 8 bytes on 32-bit architectures

execution time overhead

- system call ➪ switch to kernel mode
- locks for allocation or deallocation in a multithread context
- some system memory managers are not efficient with small objects



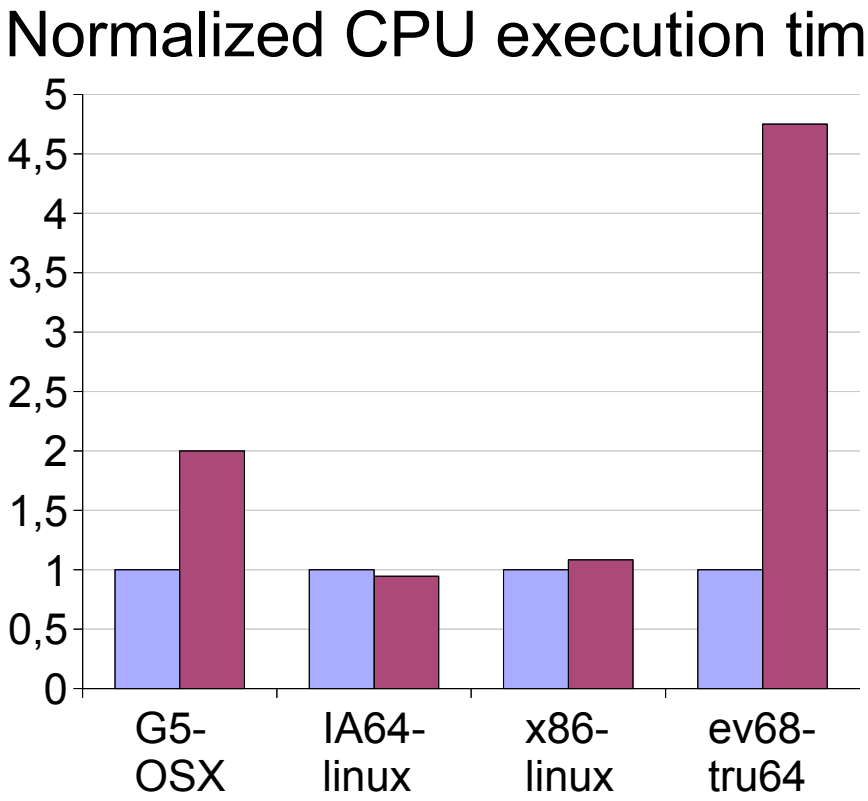
- algorithm to allocate blocks of memory (e.g., best-fit)

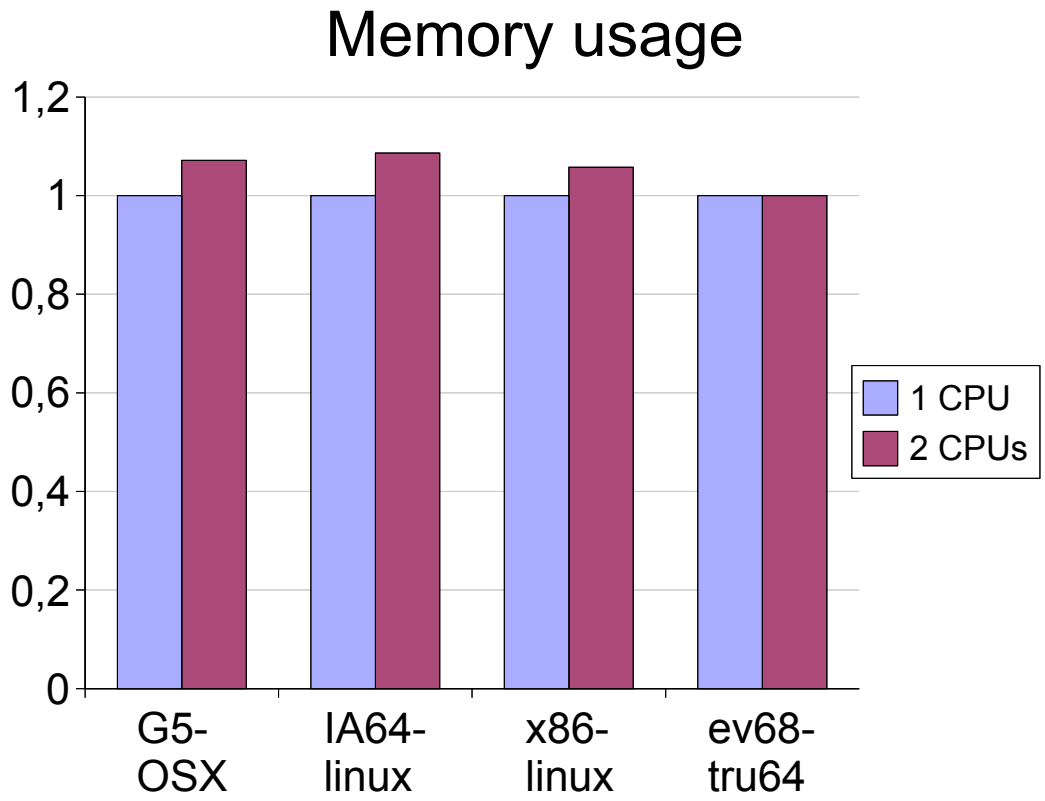
Benchmarks of the system memory manager

- Computation $V(\lambda, \lambda', \underbrace{X, Xb, Y, Yb, X', Xb', Y', Yb'}_{\text{total degree} = 12}) : 28800 \text{ terms}$

  $V^2 : \mathbf{3\ 660\ 000}$ terms

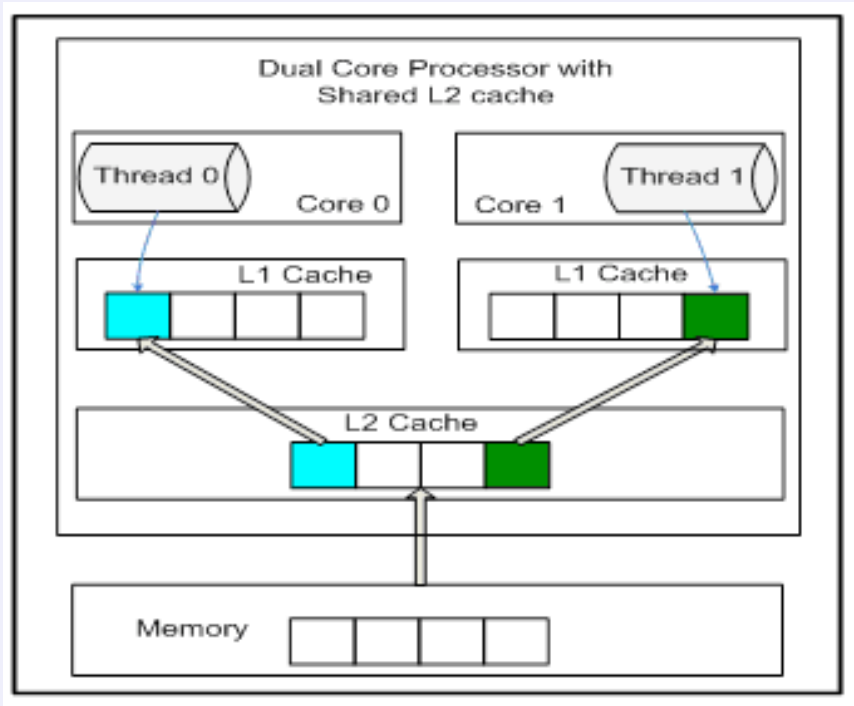
- Series stored as recursive list





Problems on multi-processor or multi-core processor

- False-sharing



- Memory contention

Optimized explicit memory manager

- Available memory managers
  - Hoard http://www.hoard.org
  - Michael Maged allocator : lock-free allocator
  - Streamflow http://people.cs.vt.edu/~scschnei/streamflow/

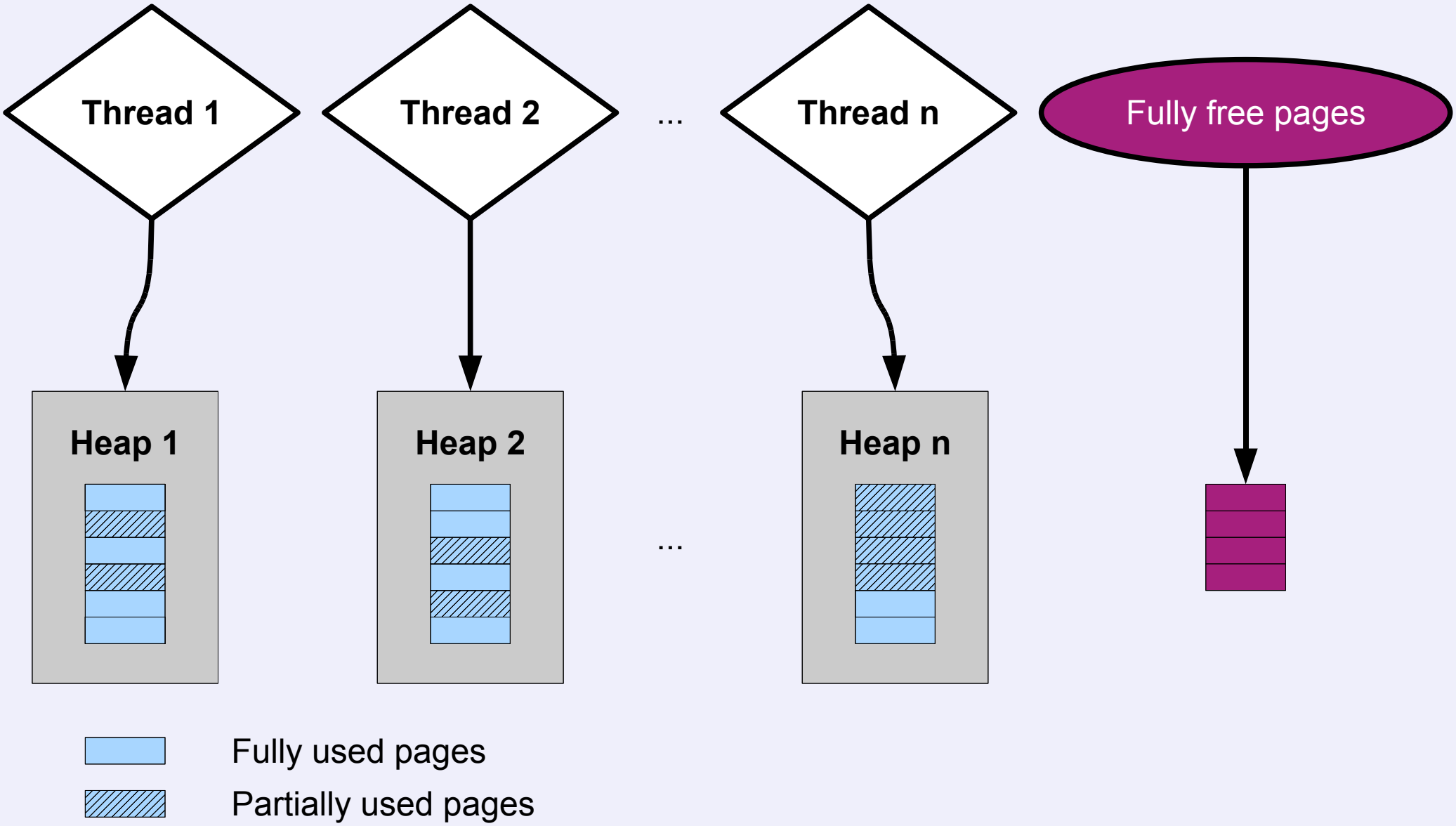
- Optimized for multi-thread context
  - avoid contention memory and false-sharing and one global list of free pages
  - small objects doesn't have header ➪ better locality

- same performance for single thread context

Allocation

- Operating systems split memory in large pages (4 or 16 or 64 Kbytes)
- For large blocks (> 4Kbytes)
  - request pages directly to the operating system kernel
- For small blocks
  - request one page and split the page in chunks of same size
  - one page contains only objects of same size ➪ only header

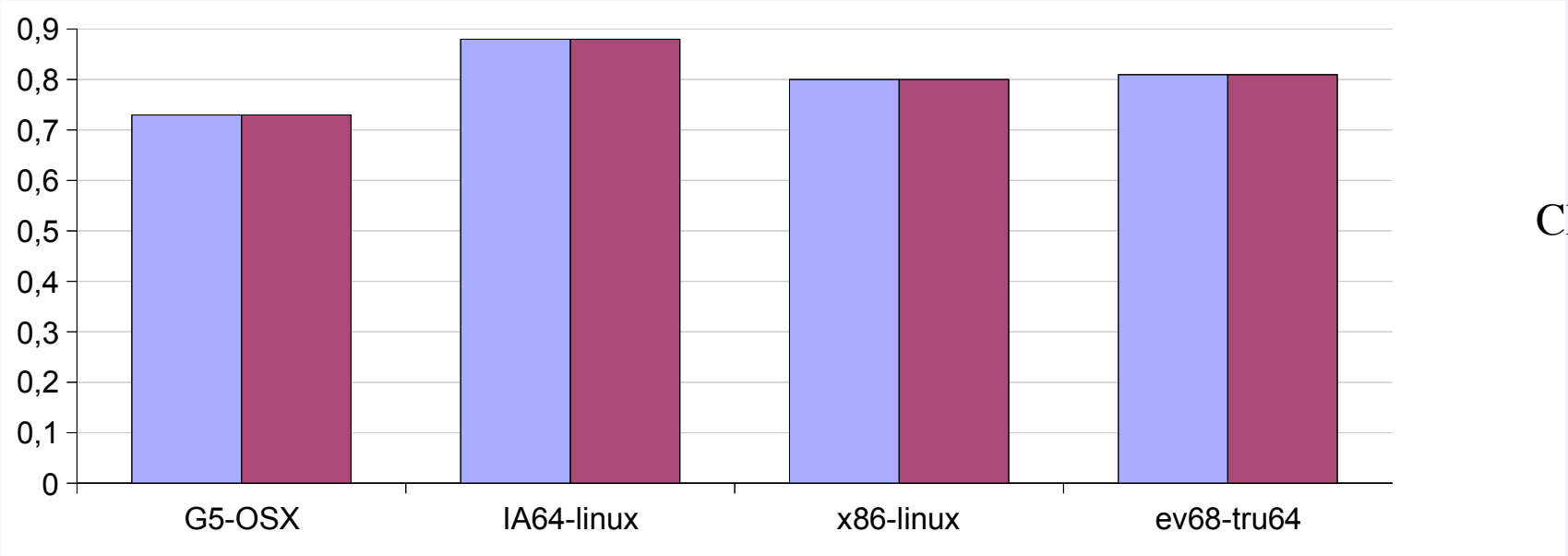
Custom memory manager for small blocks



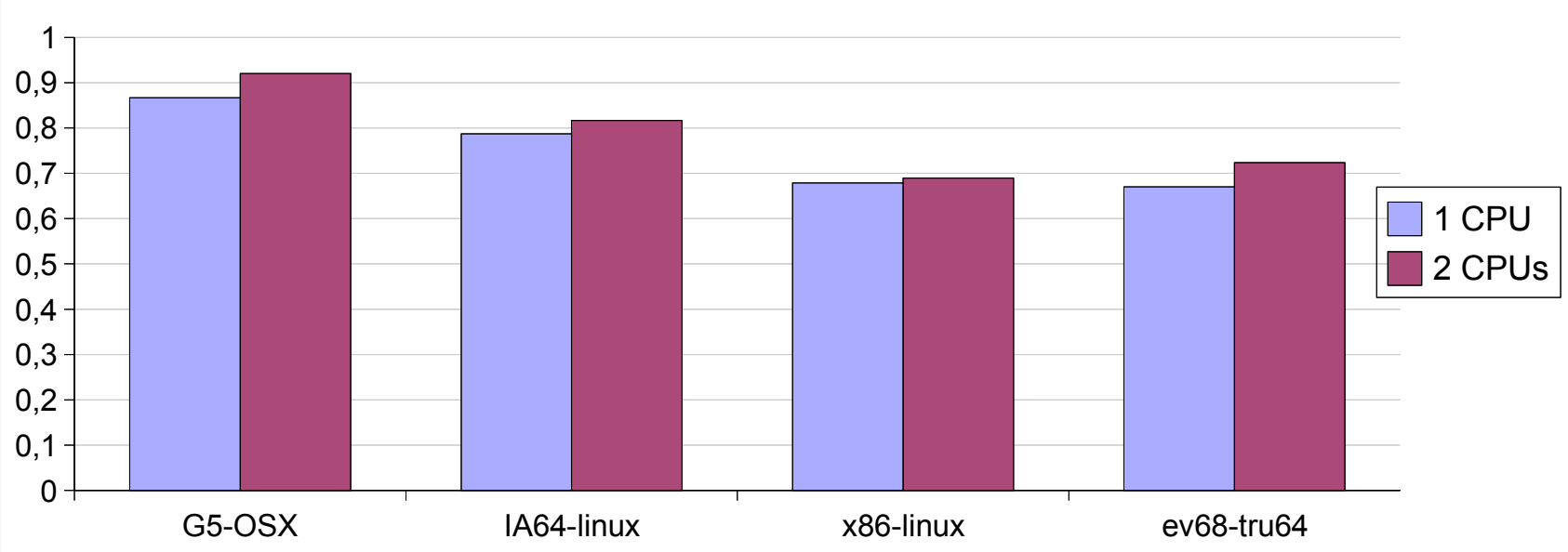
Benchmarks

- Product of 2 series (10 variables, 28800 ~> 3660000 terms)

  Normalized to the system memory manager (malloc/free)

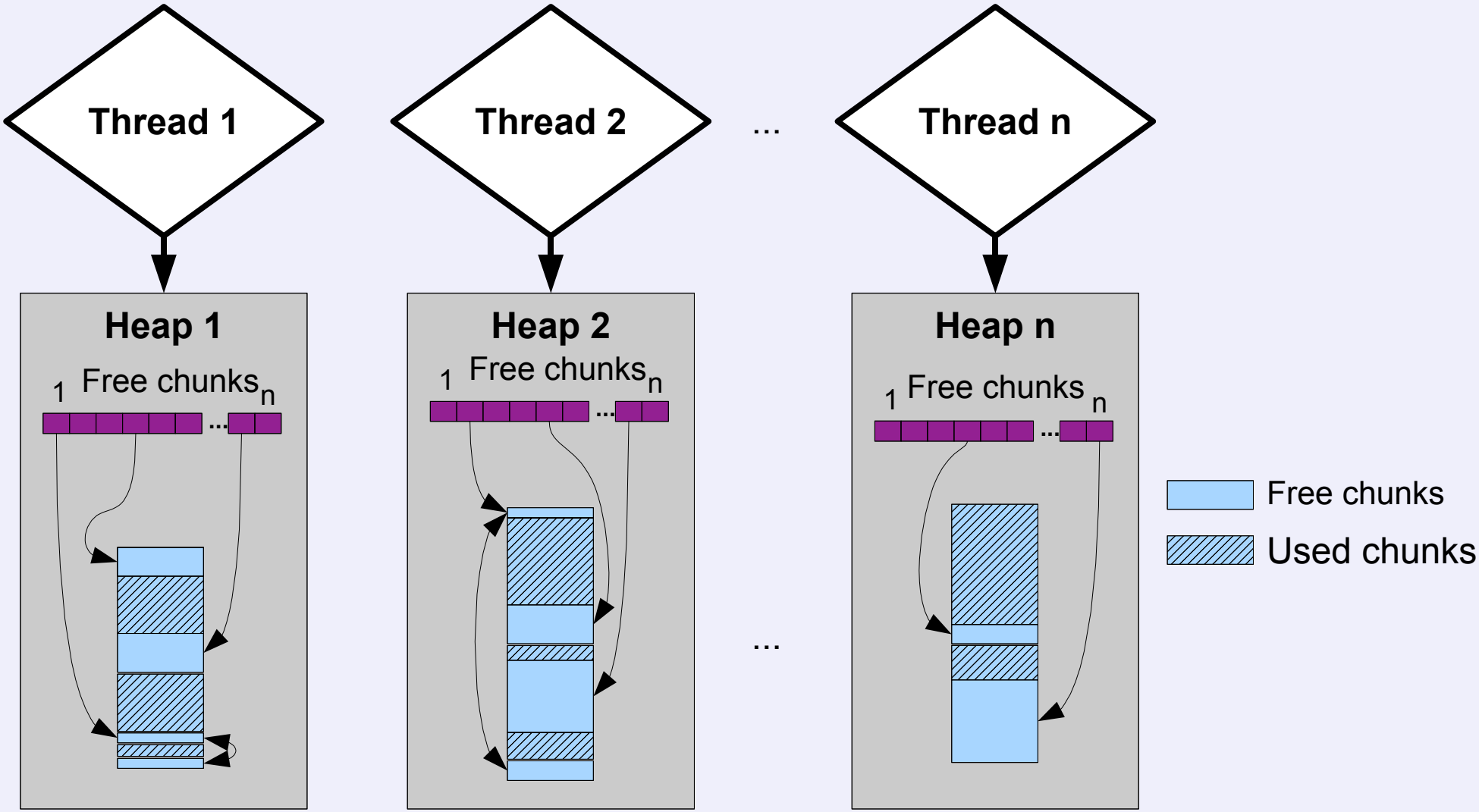


CPU execution time

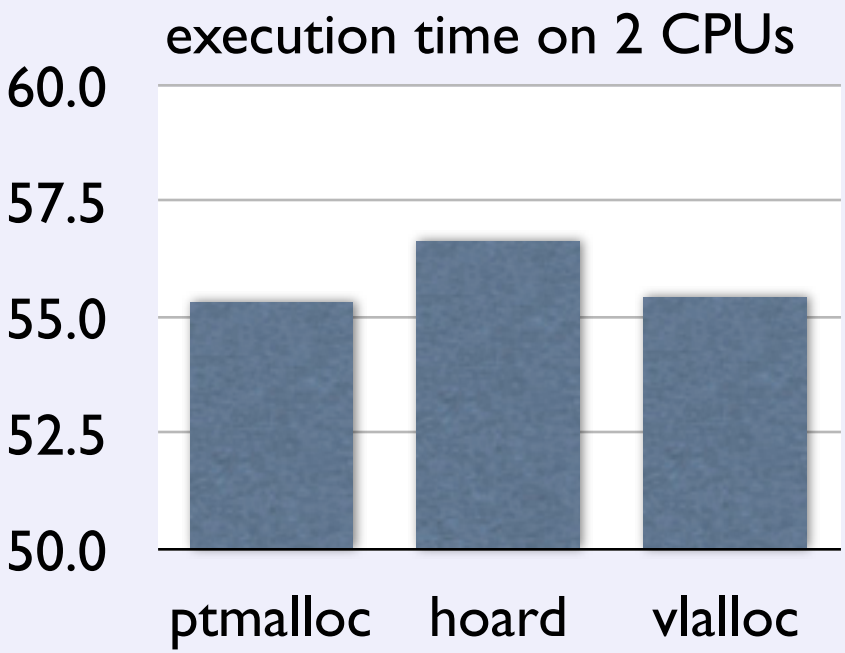


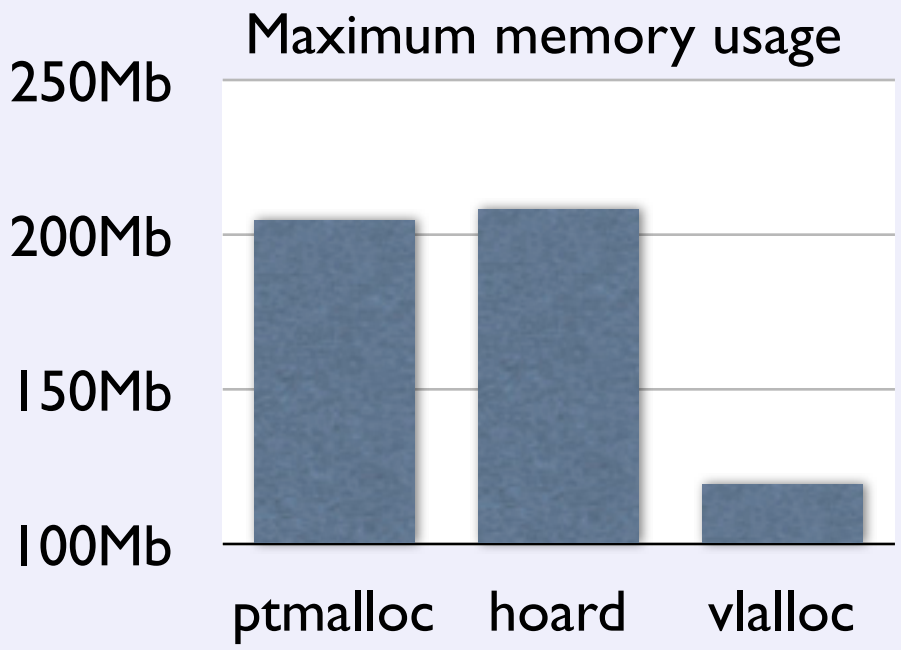
Memory usage

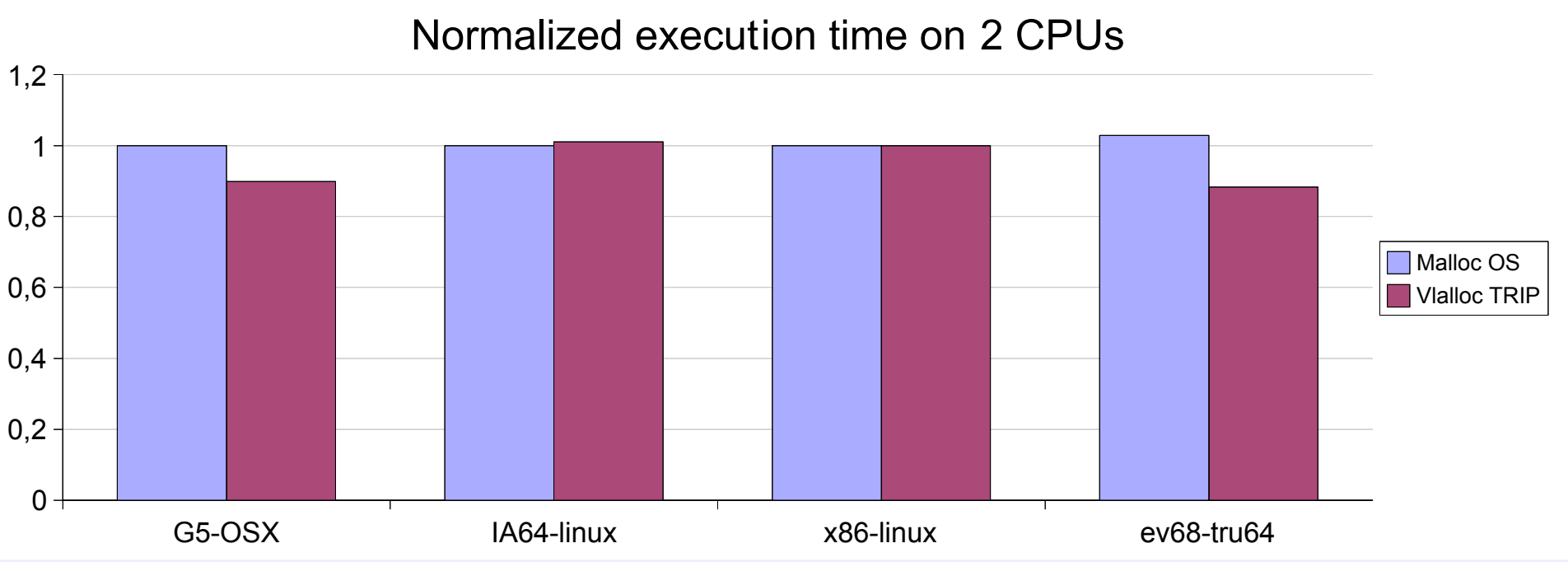
Custom memory manager for intermediate blocks



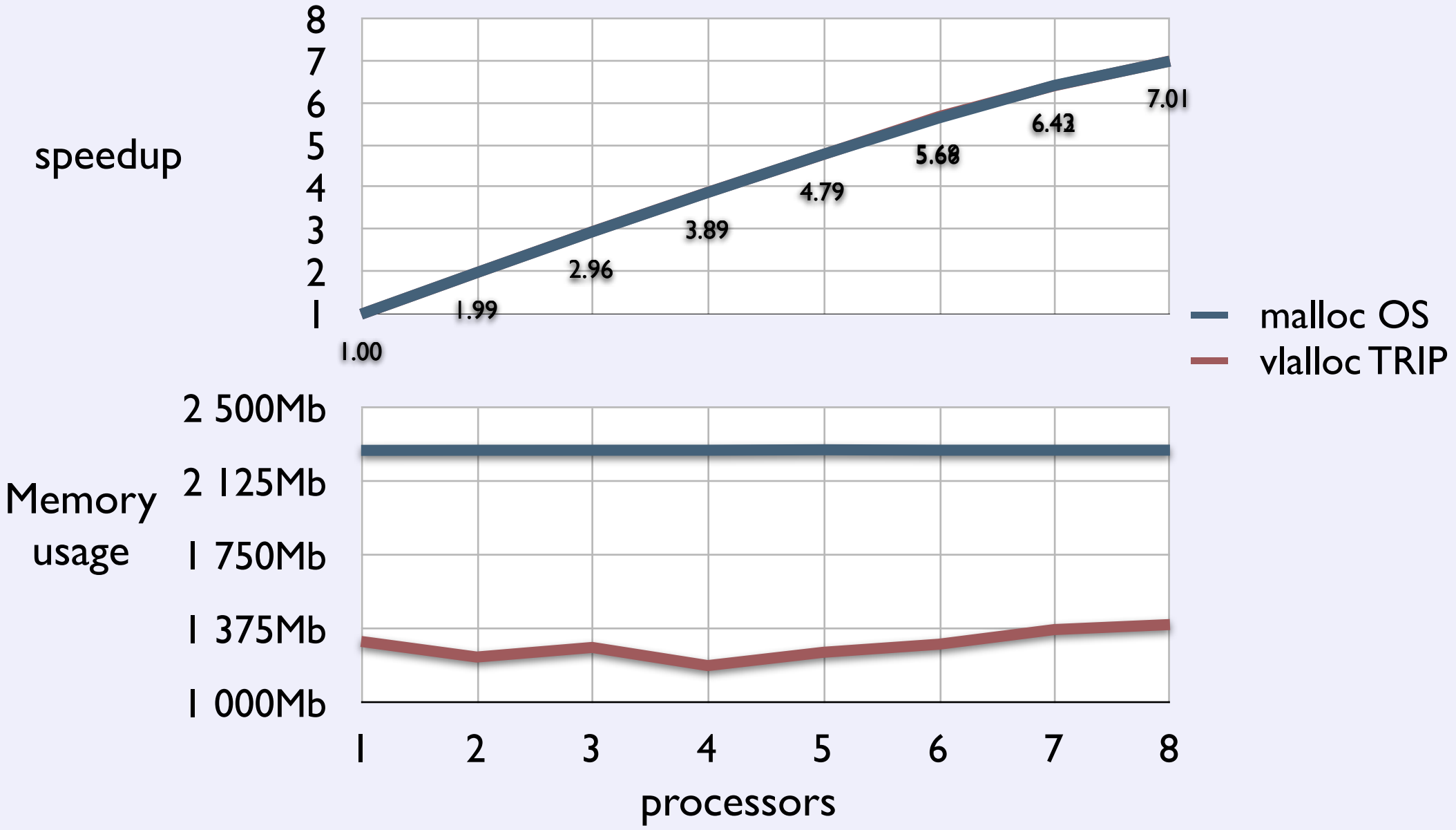
Benchmarks


execution time on 2 CPUs
60.0
57.5
55.0
52.5
50.0
ptmalloc
hoard
vlalloc


Maximum memory usage
250Mb
200Mb
150Mb
100Mb
ptmalloc
hoard
vlalloc


Normalized execution time on 2 CPUs
1,2
1
0,8
0,6
0,4
0,2
0
G5-OSX
IA64-linux
x86-linux
ev68-tru64
Malloc OS
Vlalloc TRIP

Benchmarks


speedup
8
7
6
5
4
3
2
1
1.00
1.99
2.96
3.89
4.79
5.68
6.43
7.01
malloc OS
vlalloc TRIP
Memory usage
2 500Mb
2 125Mb
1 750Mb
1 375Mb
1 000Mb
1
2
3
4
5
6
7
8
processors