Runtime

Ordonnancement de processus légers sur architectures multiprocesseurs hiérarchiques :

BubbleSched, une approche exploitant la structure du parallélisme des applications

Samuel Thibault

LaBRI

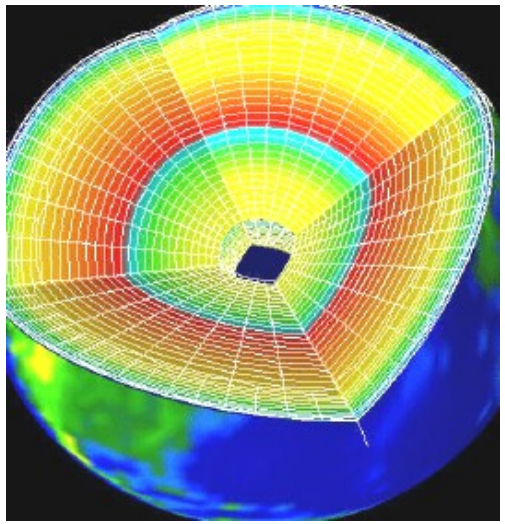
REGION AQUITAINE

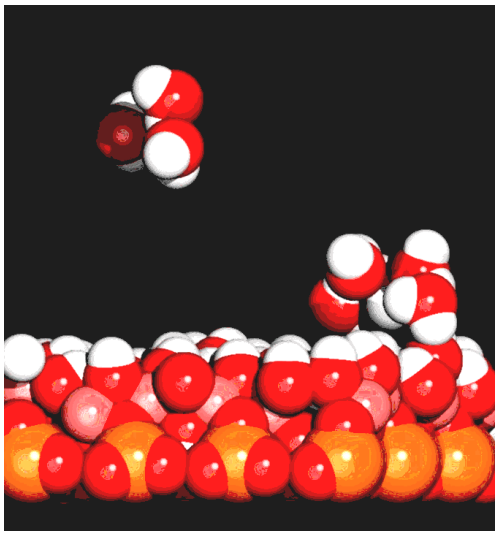
INRIA FUTURS

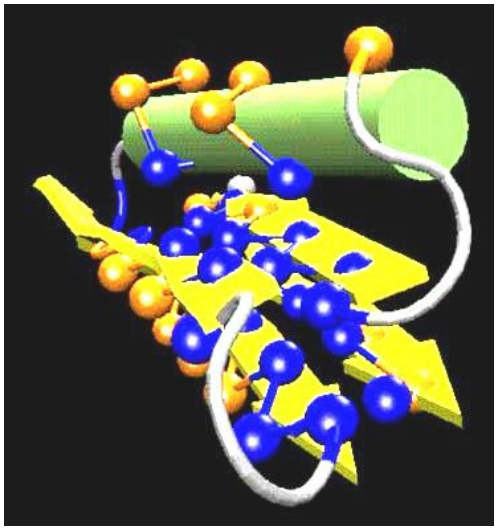
Financé par ANR

High-Performance Computing

- Simulation complements theory and experiments
  - Climatology, seismology, astrophysics, nano-sciences, material chemistry, molecular biology, ...





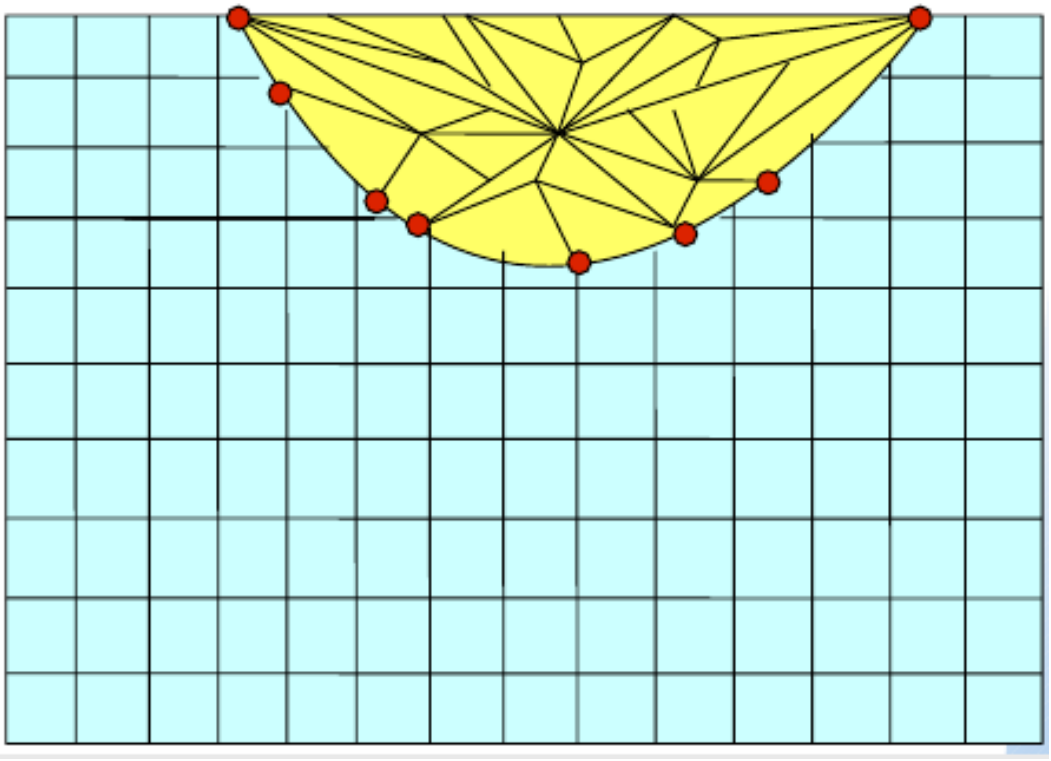


- Computation needs are always higher
  - Faster or better results

Irregular applications

- Multi-scale simulation
- Code coupling

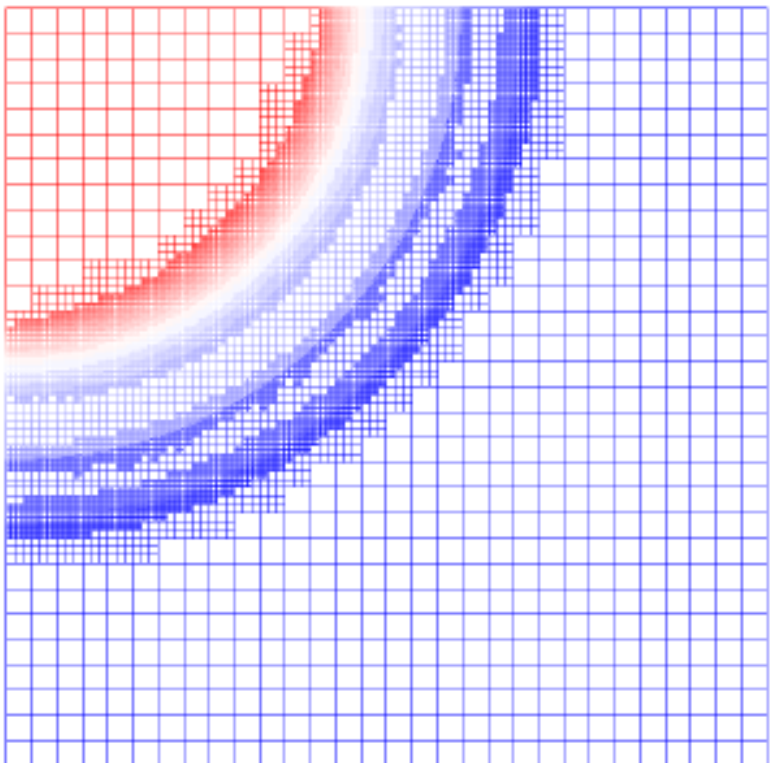
Finite Element Method

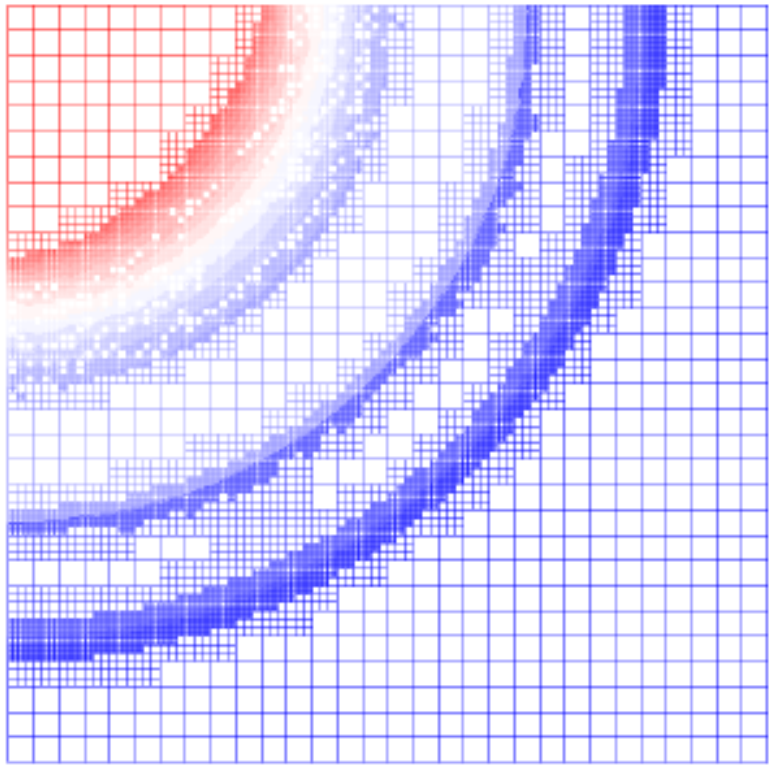


Finite Difference Method

Irregular applications

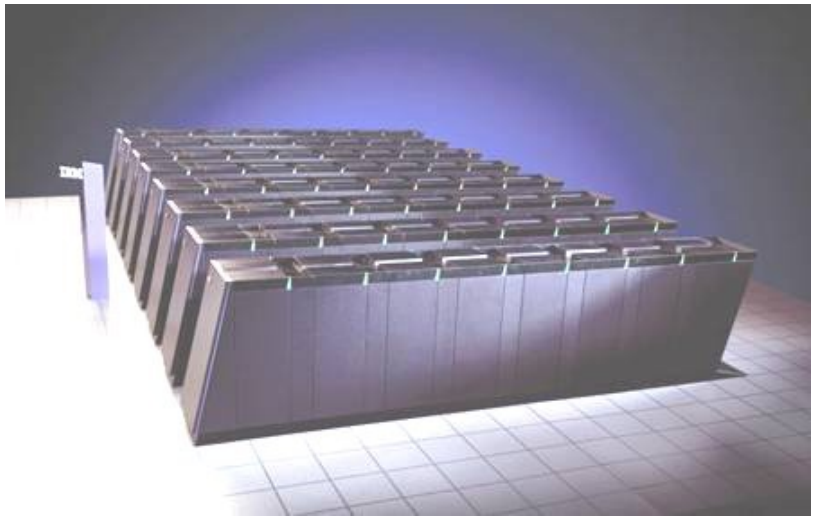
- Adaptive Mesh Refinement (AMR)
  - Behavior not known *a priori*





Towards more and more hierarchical computers

- Landscape has changed
  - From super-computers to clusters
  - With more and more parallelism

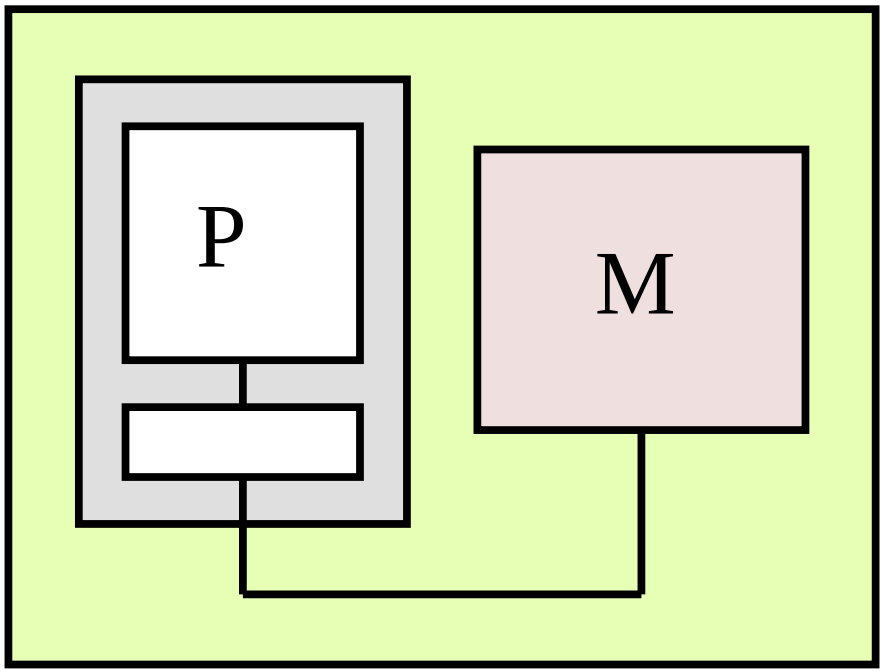


Blue Gene, 106,496 cores

Tera10, 8704 cores

Towards more and more hierarchical computers

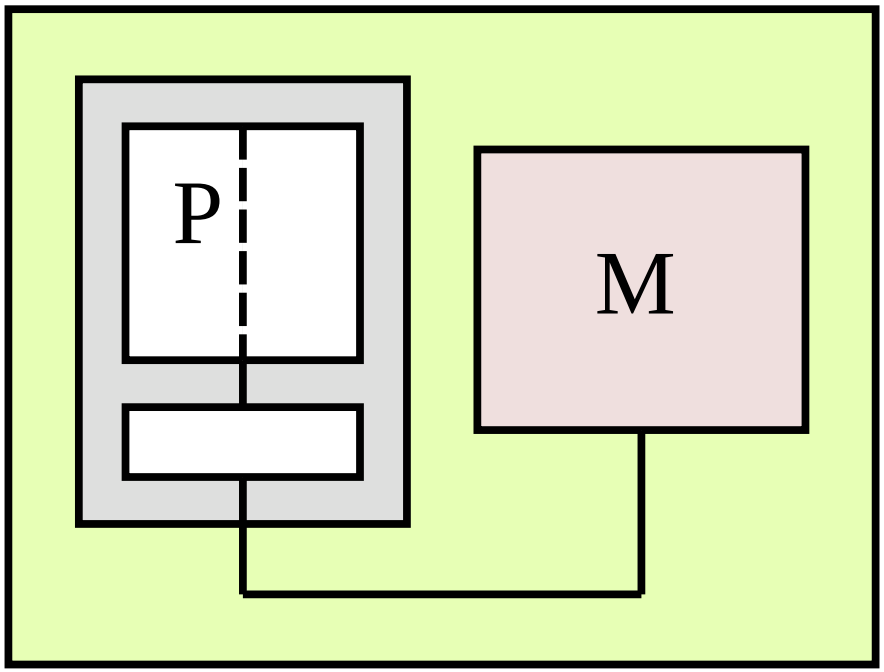
Single processor



Towards more and more hierarchical computers

- Simultaneous MultiThreading

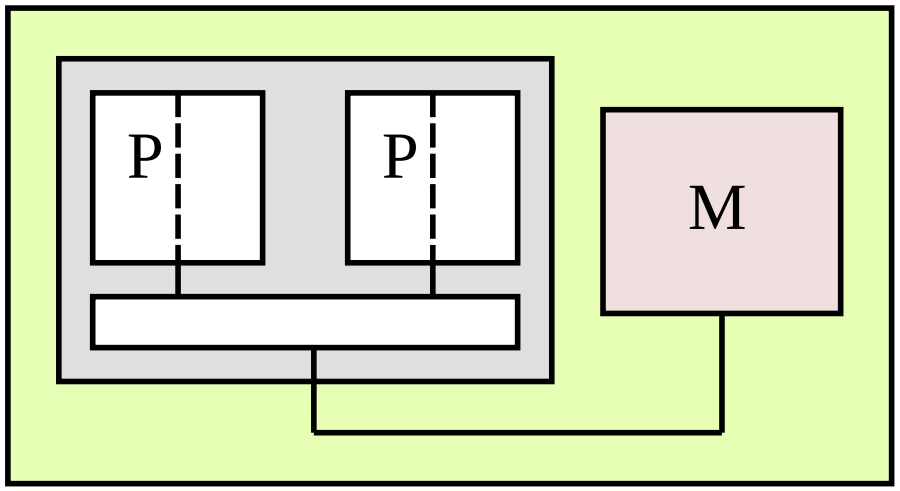
  (HyperThreading)



Towards more and more hierarchical computers

- SMT

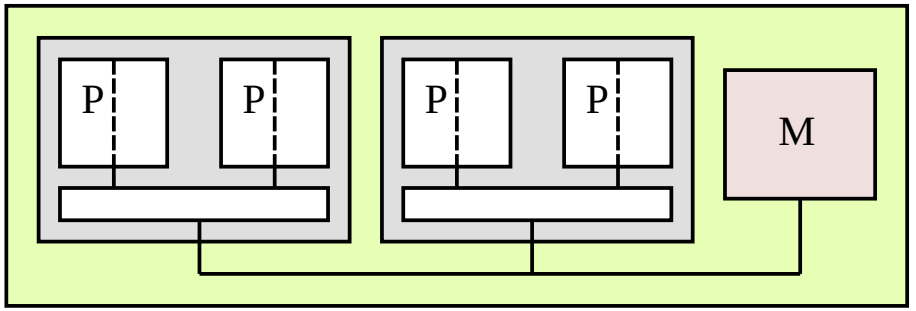
  (HyperThreading)
- Multi-Core



Towards more and more hierarchical computers

- SMT

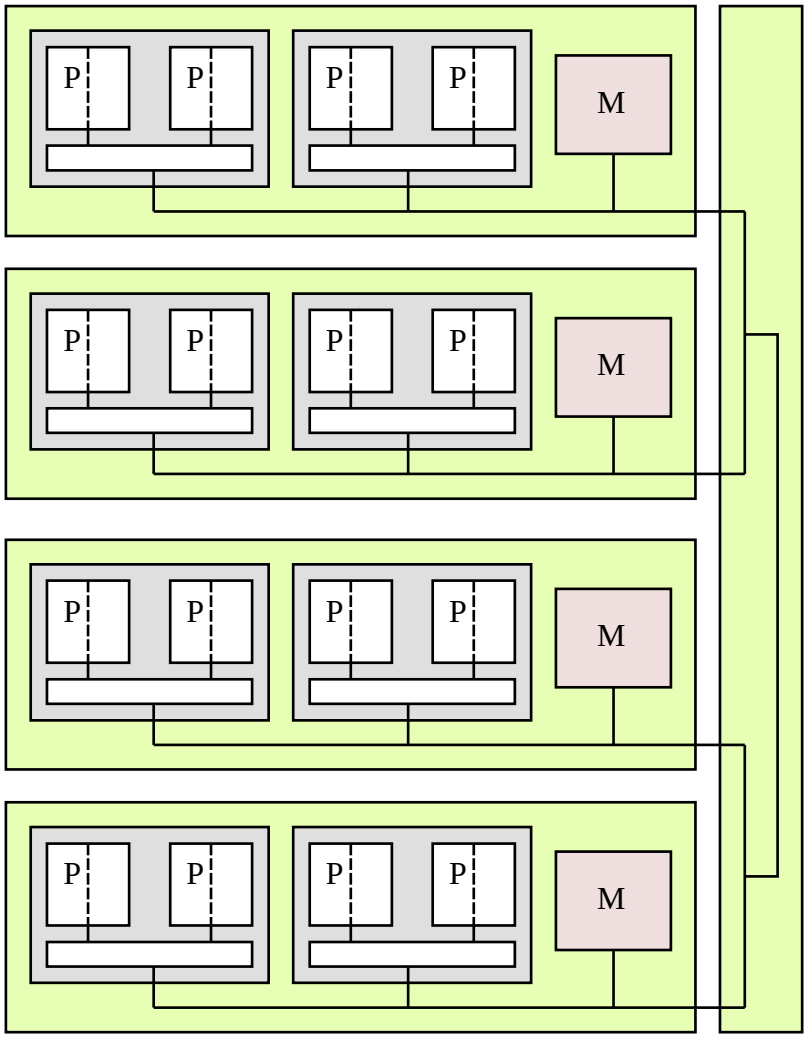
  (HyperThreading)
- Multi-Core
- Symmetric Multi-Processor



Towards more and more hierarchical computers

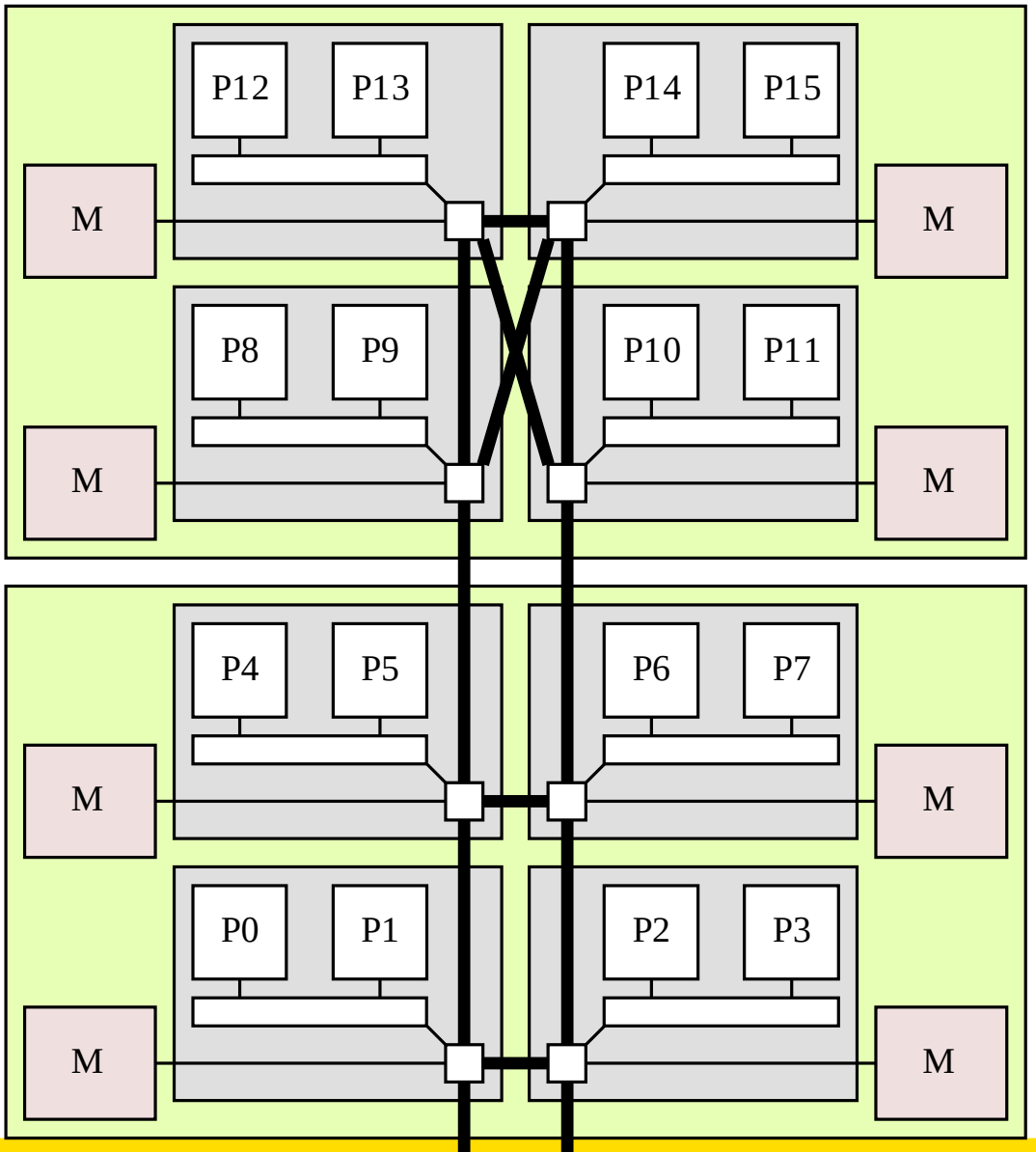
- SMT

  (HyperThreading)
- Multi-Core
- SMP
- Non-Uniform Memory Access (NUMA)



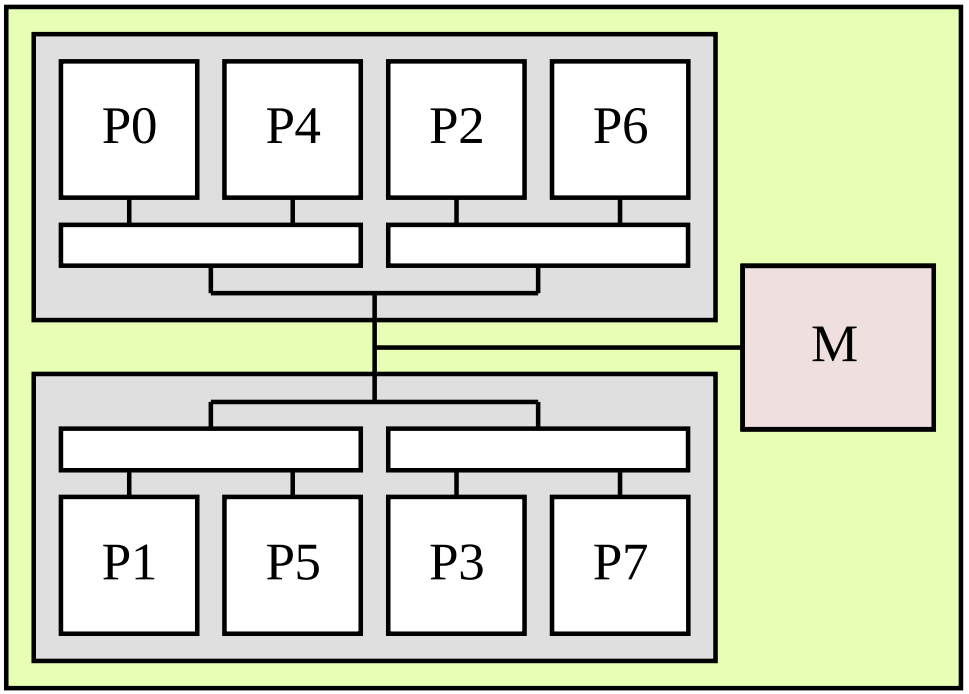
Hagrid, octo-dual-core

- AMD Opteron
- NUMA factor 1.1-1.5



Aragog, dual-quad-core

- Intel
- Hierarchical cache levels



How to run applications on such machines?

How to program parallel machines?

- By hand
  - Tasks, POSIX threads, explicit context switch
- High-level languages
  - Processes, task description, OpenMP, HPF, UPC, ...
- Technically speaking, threads
- How to schedule them efficiently?

How to schedule efficiently?

- Performance
  - Affinities between threads and memory taken into account
- Flexibility
  - Execution easily guided by applications
- Portability
  - Applications adapted to any new machine

Predetermined approaches

- Two phases
  - Preliminary computation of
    - Data placement [Marather, Mueller, 06]
    - Thread scheduling
  - Execution
    - Strictly follows the pre-computation
- Example: PaStiX [Hénon, Ramet, Roman, 00]

✔ Excellent performances

✗ Not always sufficient or possible: strongly irregular problems...

Opportunistic approaches

- Various greedy algorithms
  - Single / several [Markatos, Leblanc, 94] /
    a hierarchy of task lists [Wang, Wang, Chang, 00]
- Used in nowaday's operating systems
  - Linux, BSD, Solaris, Windows, ...

✔ Good portability

✗ Uneven performances

  - No affinity information...

Negotiated approaches

- Language extensions
  - OpenMP, HPF, UPC, ...

✔ Portability (adapts itself to the machine)

✗ Limited expressivity (e.g. no NUMA support)

- Operating System extensions
  - NSG, liblgroup, libnuma, ...

✔ Freedom for programmers

✗ Static placement, requires rewriting placement strategies according to the architecture

Issues

- Negotiated approach seems promising, but
  - Which scheduling strategy?
    - Depends on the application
  - Which information to take into account?
    - Affinities between threads?
    - Memory occupation?
  - Where does the runtime play a role?
- But there is hope!
  - Programmers and compilers do have some clues to give
  - Missing piece: structures

BubbleSched

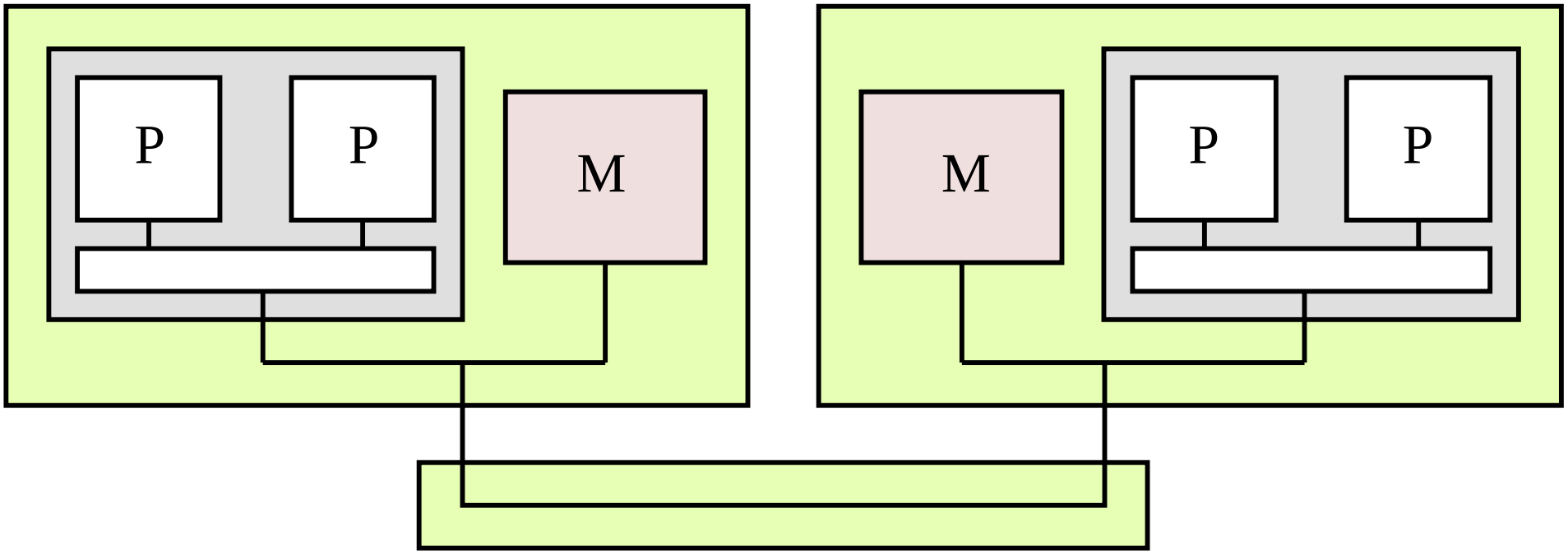
Guiding scheduling through bubbles

Idea: Structure to better schedule

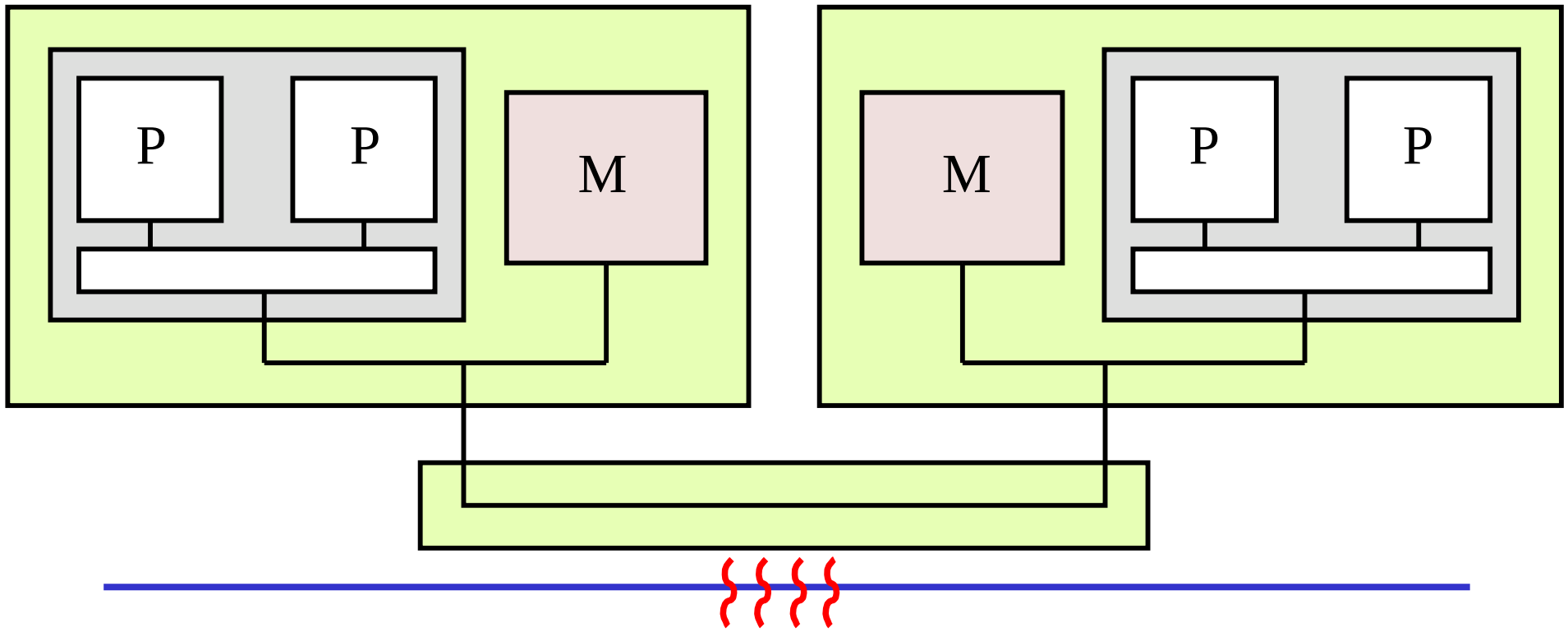
Bridging the gap between programmers and architectures

- Grab the structure of the parallelism
  - Express relations between threads, memory, I/O, ...
- Model the architecture in a generic way
  - Express the structure of the computation power
- Scheduling is mapping
  - As it should just be!
  - Completely algorithmic
  - Allows all kinds of scheduling approaches

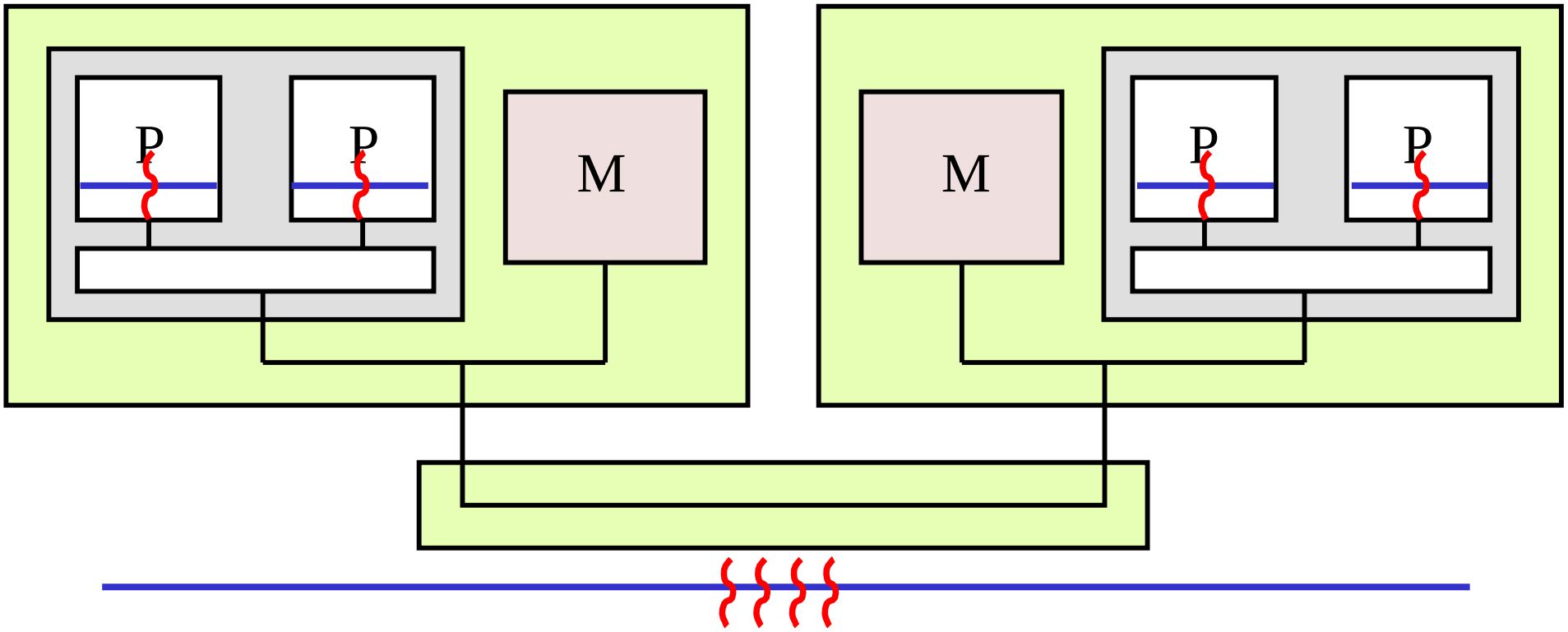
Runqueues to model hierarchical machines



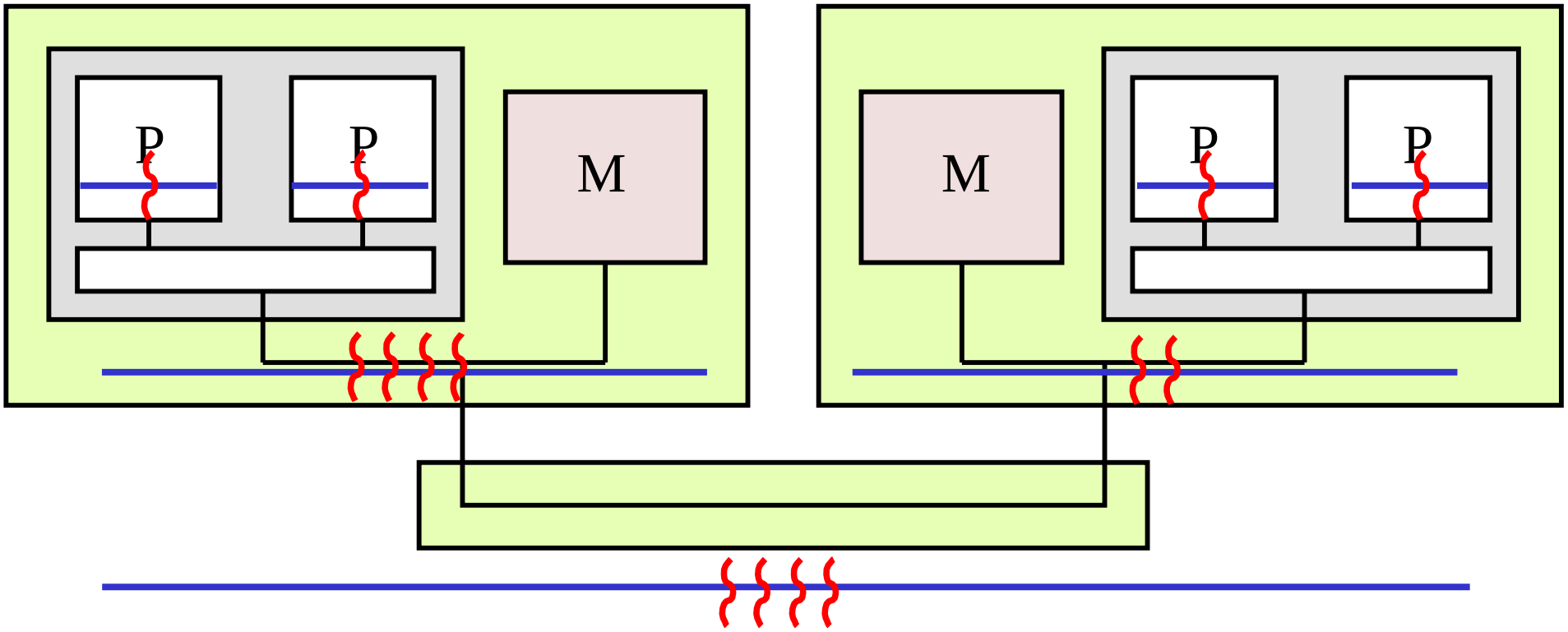
Runqueues to model hierarchical machines



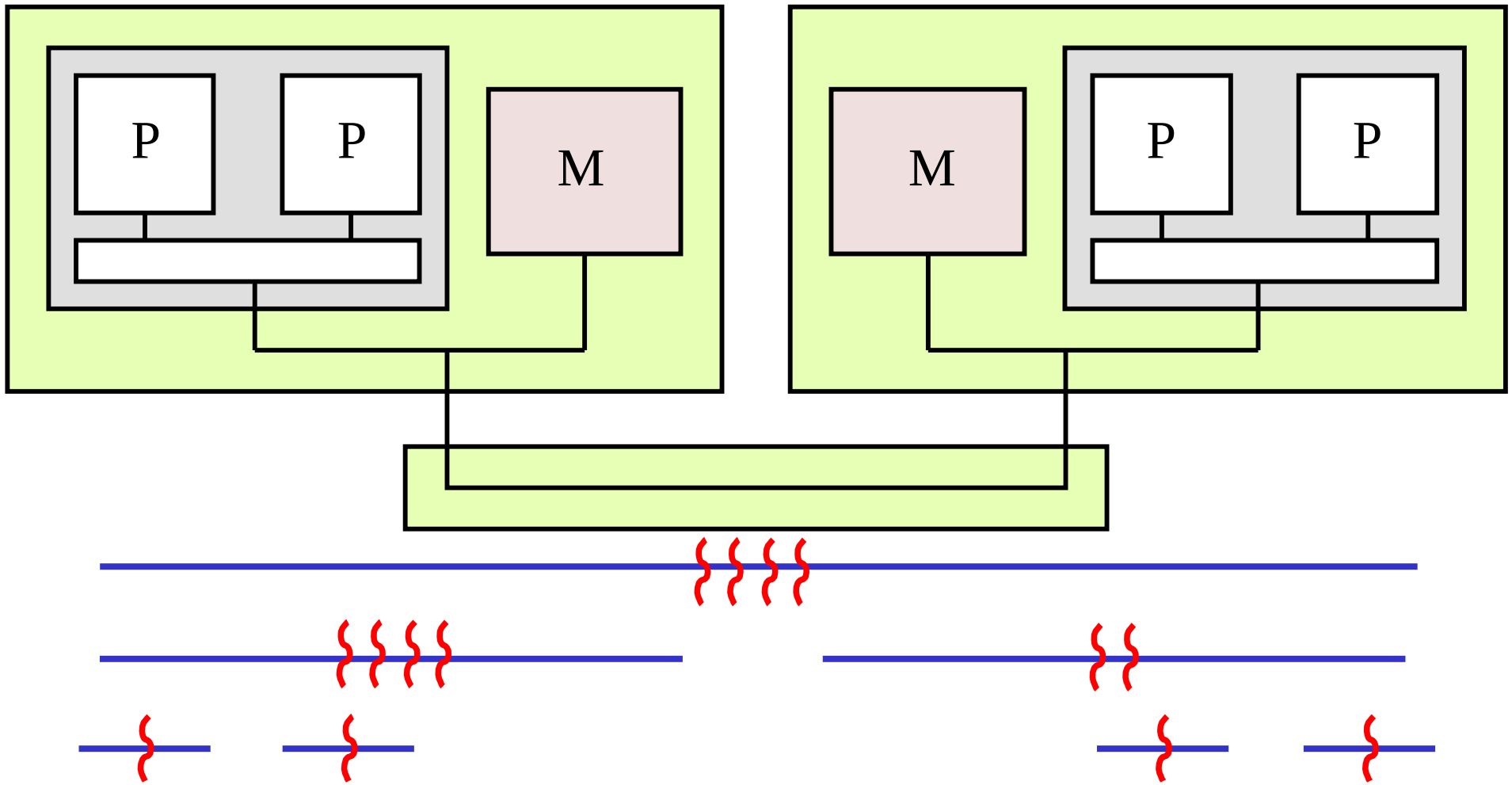
Runqueues to model hierarchical machines



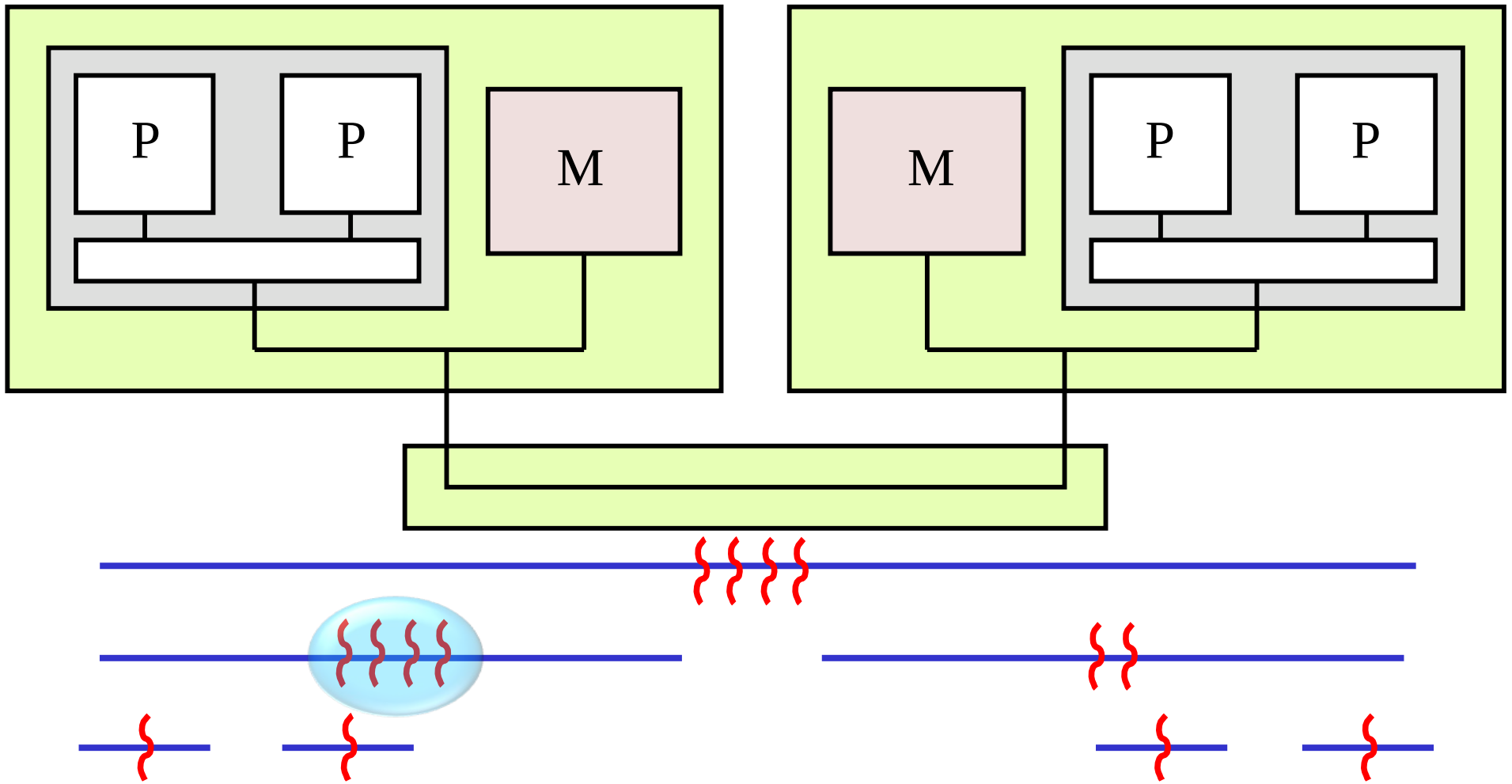
Runqueues to model hierarchical machines



Runqueues to model hierarchical machines



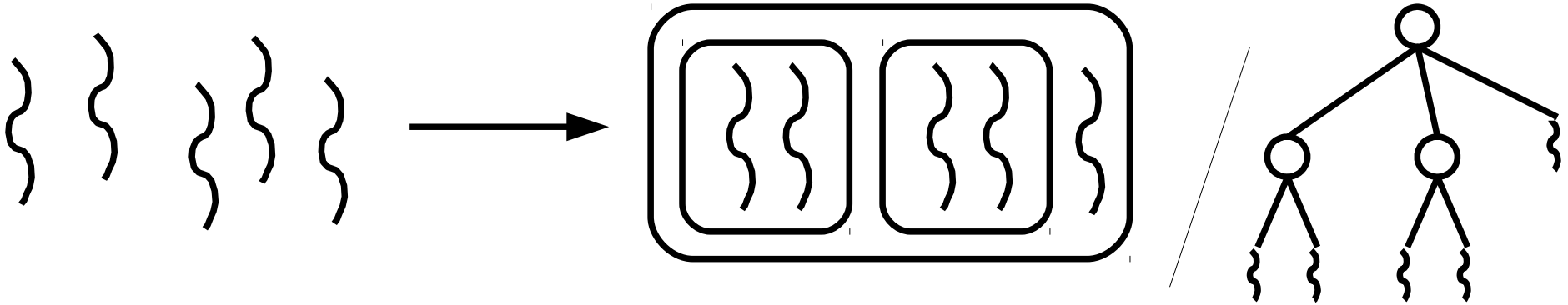
Runqueues to model hierarchical machines



Bubbles to model thread affinities

Keeping the structure of the application in mind

- Data sharing
- Collective operations
- ...

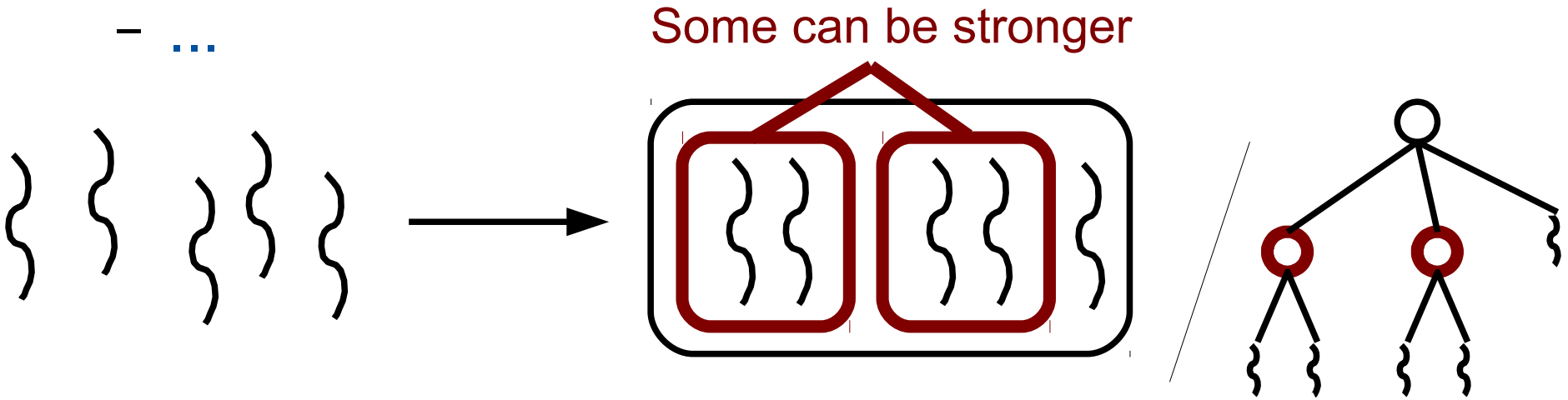


```
bubble_insert_thread(bubble, thread);
bubble_insert_bubble(bubble, subbubble);
```

Bubbles to model thread affinities

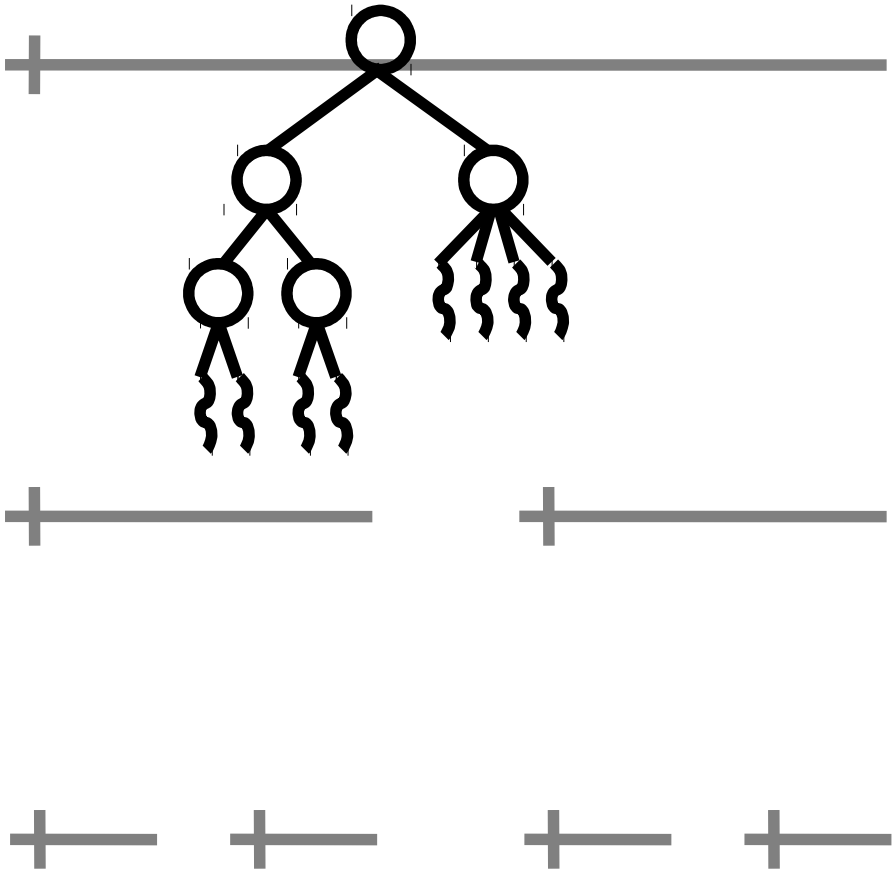
Keeping the structure of the application in mind

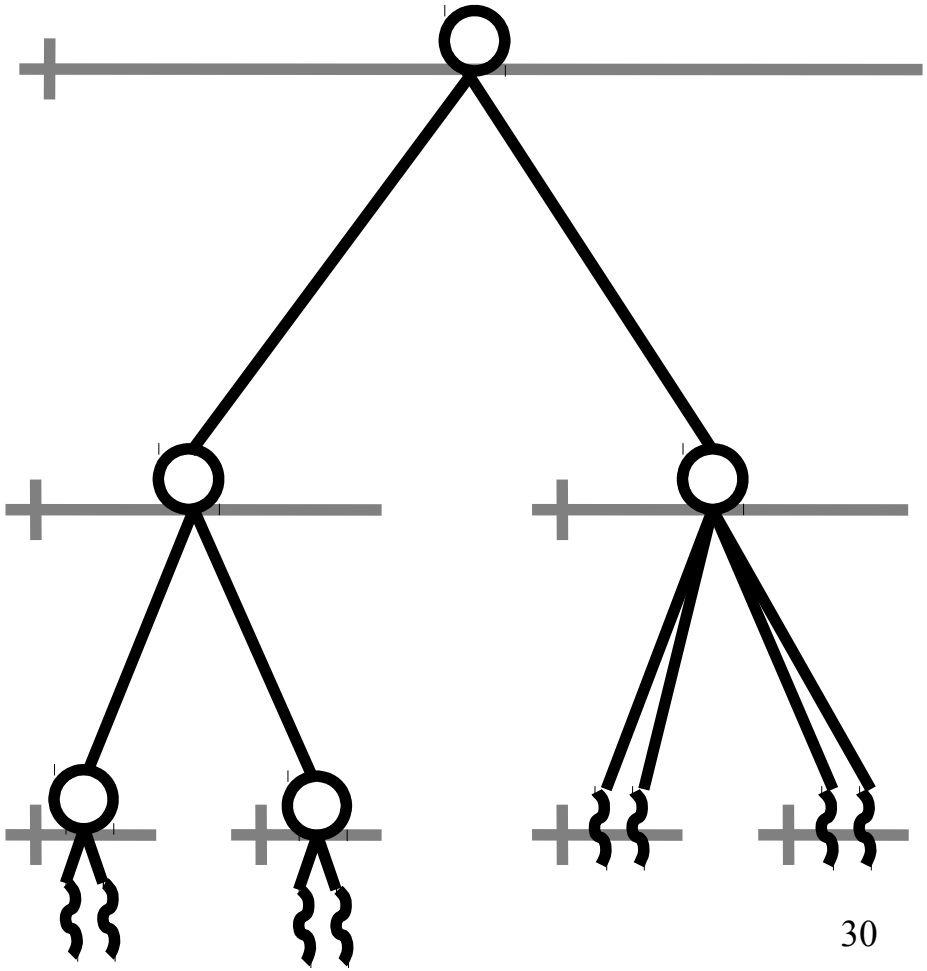
- Data sharing
- Collective operations
- ...




```
bubble_insert_thread(bubble, thread);
bubble_insert_bubble(bubble, subbubble);
```

Examples of thread and bubble repartitions





A lot of useful information linked with bubbles

- From the hardware
  - Target machine architecture
  - Performance counters
- From the compiler
  - Data access pattern
  - Data amount
- From the programmer
  - Threads and thread / data affinities
  - Thread behavior: I/O vs CPU

Various applications, various schedulers

- Various behaviors
  - Memory affinity → keep on the same NUMA node or even same chip
  - Memory bandwidth → distribute over chips
  - Irregular parallelism → keep threads “up”

➔ Compromises have to be found

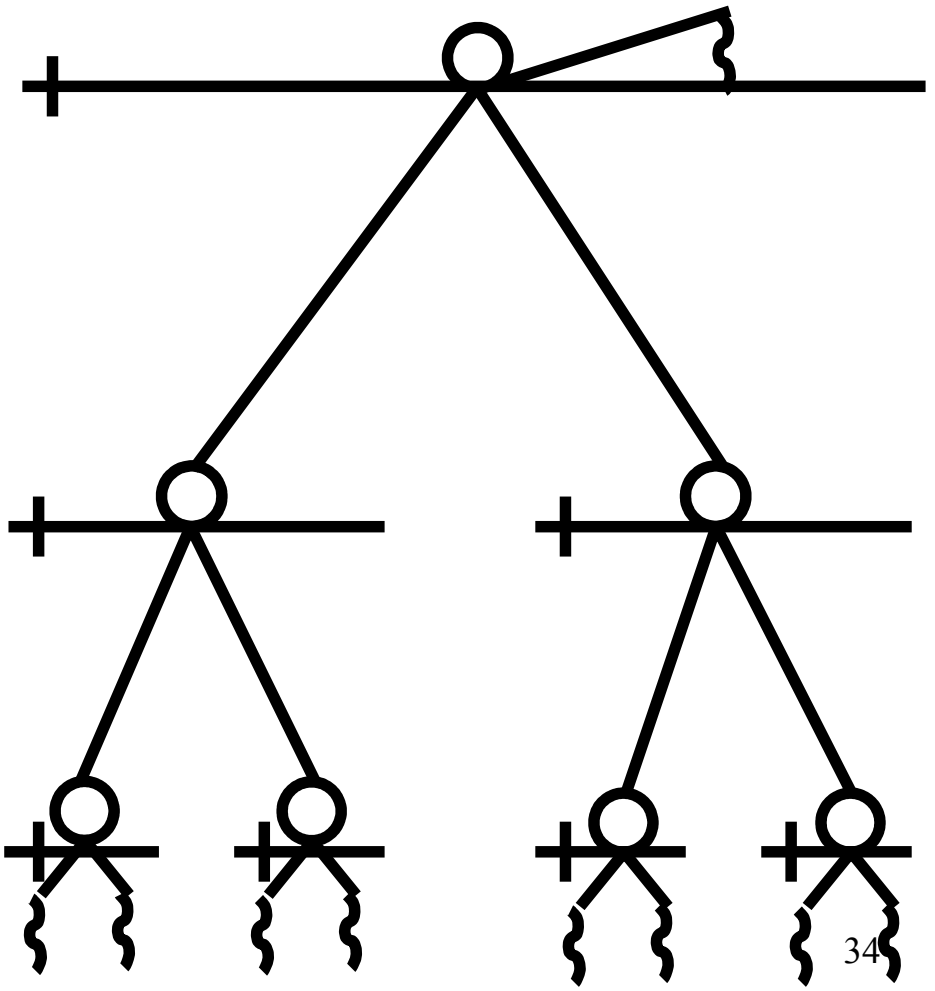
➔ No generic scheduler can fit all situations

A 3-side Approach

- Application programmer
  - Uses high-level language (OpenMP, HPF, UPC, ...)
  - Handles application development
- Scheduling programmer
  - Uses high-level scheduling primitives
  - Handles scheduling algorithms
- Technical programmer
  - Implements scheduling primitives
  - Handles technical details

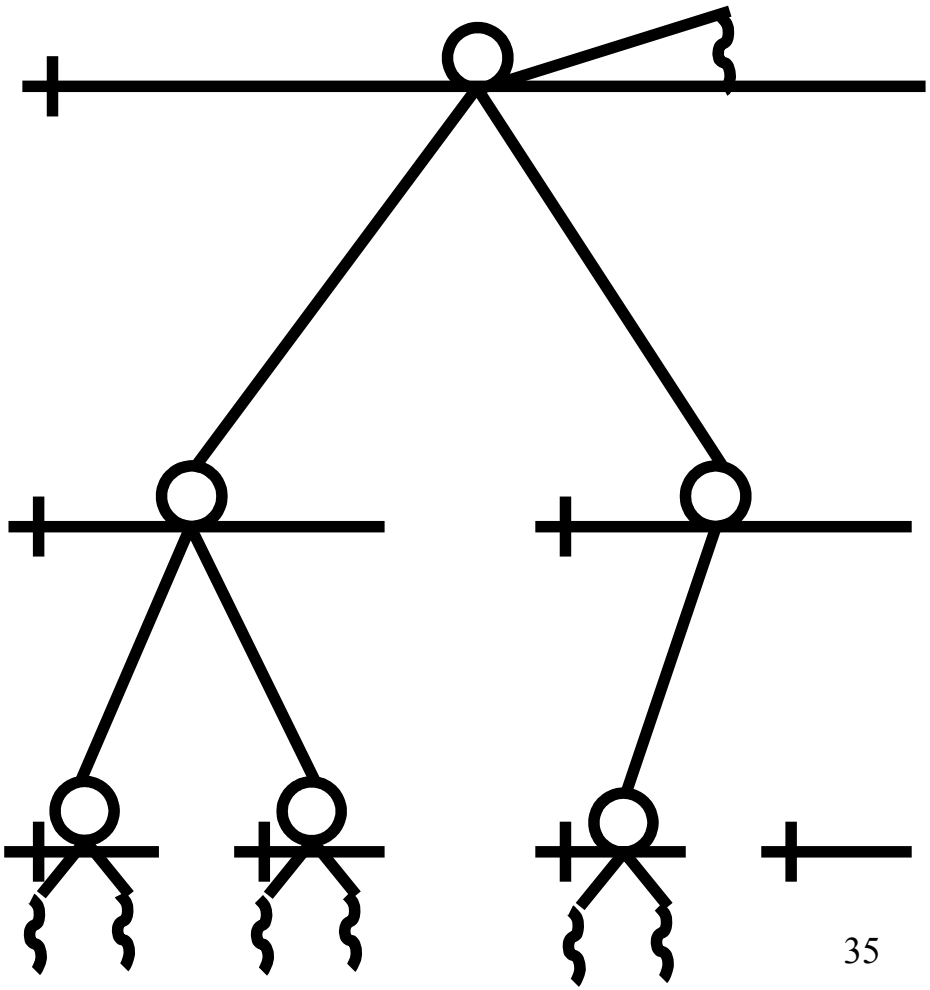
Toolbox for distributing threads and bubbles

- Walk the machine
  - rq->father, rq->children[]
- Look
  - rq_for_each_entry
- Lock
  - rq_lock, rq_unlock
  - all_lock, all_unlock
- Move
  - get_entity, put_entity



Toolbox for distributing threads and bubbles

- Hooks
  - Idle
  - Timeslice
  - Bubble/thread creation
  - ...
- Dedicated thread
  - « daemon »

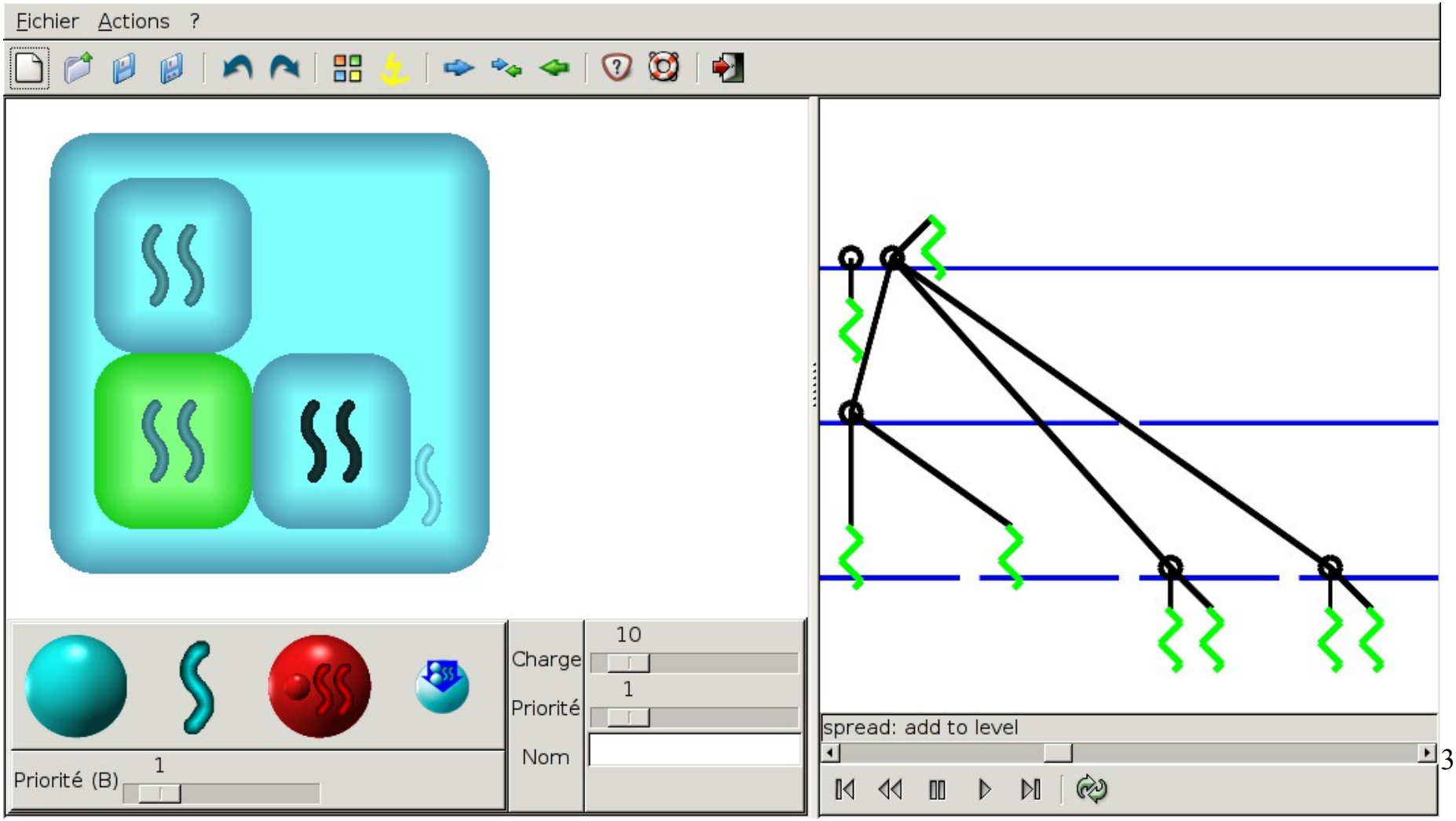


Understanding performances

- Generate a trace of events during execution
  - FxT library, co-developped with UNH
- Convert into Flash animation
- Graphical debugger

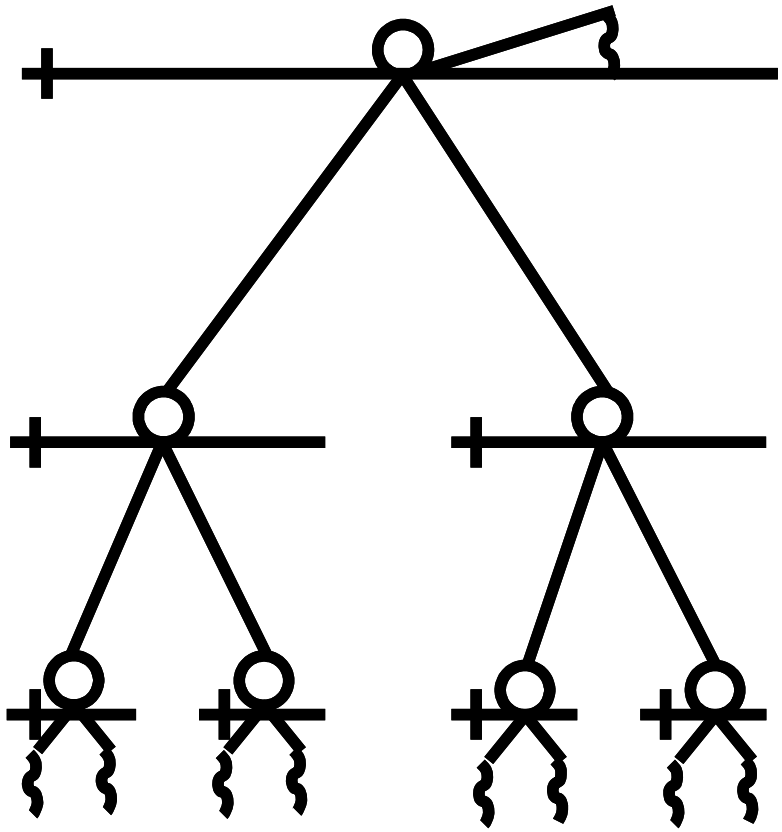
Click & Play Interface

Quickly test synthetic bubble hierarchies



Implemented schedulers

- Full-featured schedulers
  - Gang scheduling
  - Spread
    - Favor load balancing
  - Affinity
    - Favor affinities (Broquedis)
    - Memory aware (Jeuland)
- Reuse and compose
  - Work stealing
  - Combined schedulers (time, space, etc.)



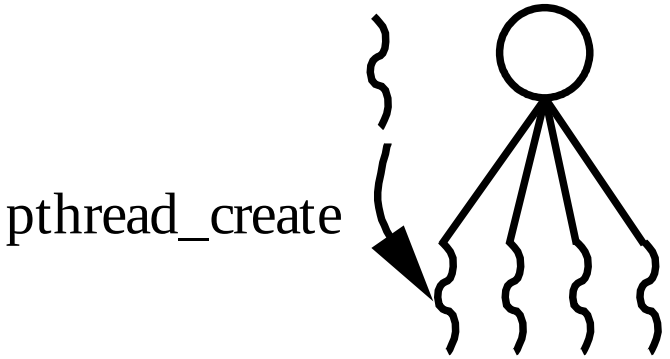
Implementation within Marcel

- User-level thread scheduler
- Efficient, flexible and portable
- No need to patch the kernel

(µs)	Creation	Execution
Marcel mono	0,60	0,46
Marcel SMP	0,80	0,60
Marcel NUMA	0,85	0,60
Marcel NUMA bubbles	0,85	0,70
Linux	6,4	11,3

Automatic bubble generation

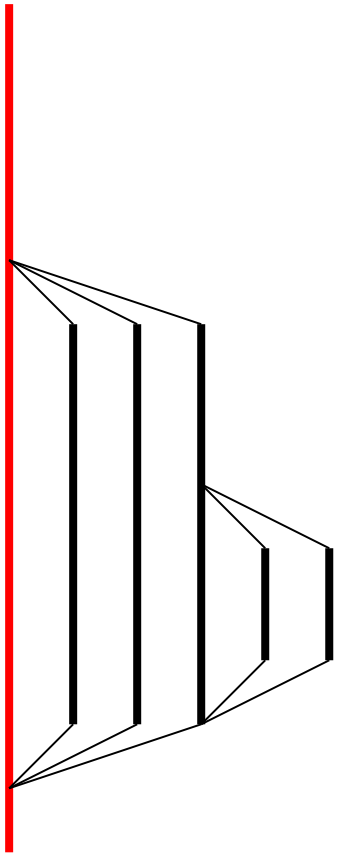
- POSIX compatible, NPTL ABI compatible
  - Father-children relationship

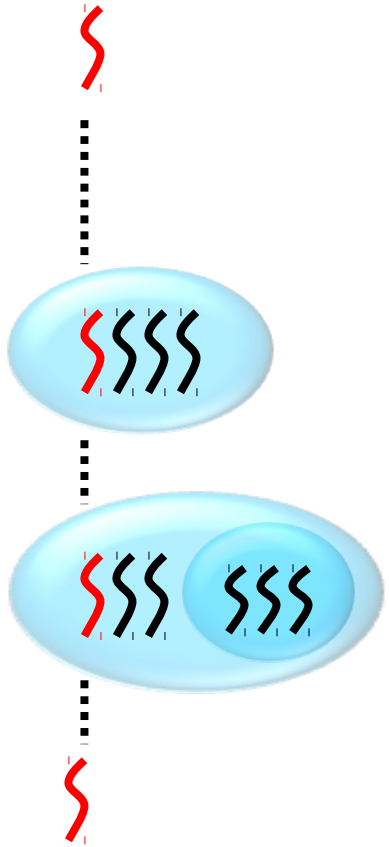


- GCC OpenMP backend

Building bubbles out of OpenMP directives

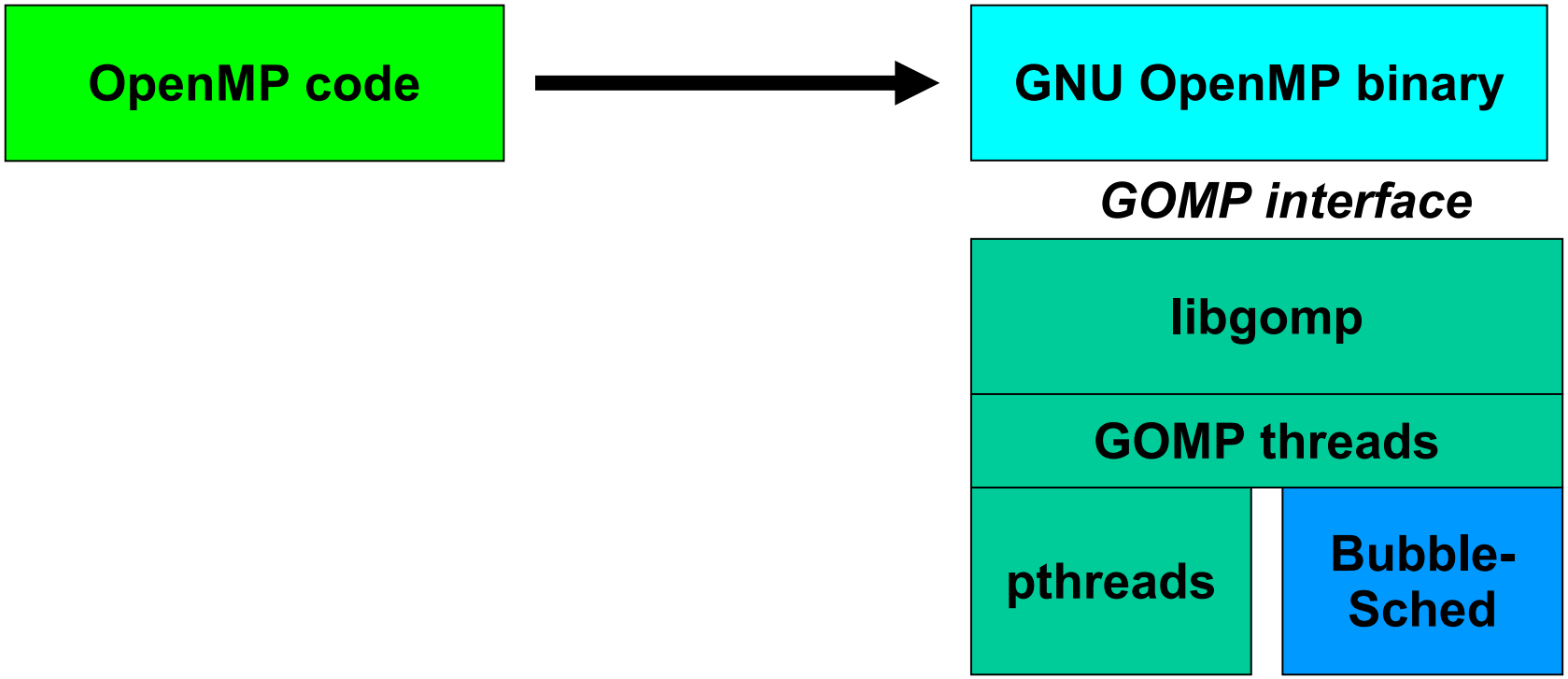
- Create one bubble per parallel section
  - Including for nested parallel sections





Building bubbles out of OpenMP directives

- GNU OpenMP port on Marcel threads + bubbles
- Binary compatibility with existing applications



Case studies

Case studies

- SuperLU
  - Posix
  - Task parallelism
- BT-MZ
  - OpenMP
  - Space-irregular parallelism
- MPU
  - OpenMP
  - Time-irregular parallelism

Task parallelism SuperLU

- Irregular demand for LU factorization jobs
- Efficient parallel routine (SuperLU)

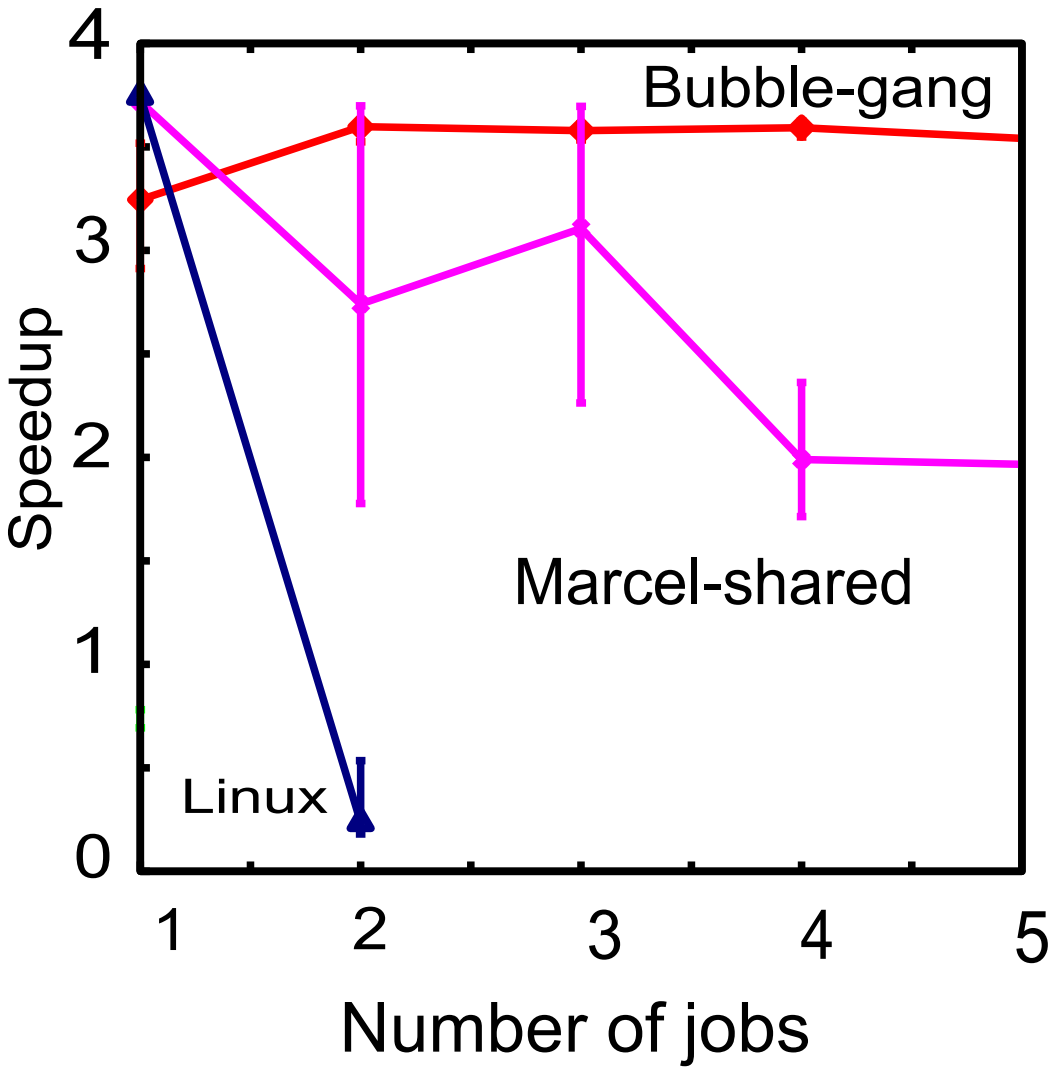
- Dual-dual core Opteron (hence 4 CPUs)

➔ Gang scheduling

A gang scheduler

```
void *gang_sched(void *param) {
    while(1) {
        rq_lock(&main_rq);
        rq_lock(&nosched_rq);
        rq_for_each_entry(&main_rq, &e) {
            get_entity(e, &main_rq);
            put_entity(e, &nosched_rq);
        }
        if (!rq_empty(&nosched_rq)) {
            e = rq_entry(&nosched_rq);
            get_entity(e, &nosched_rq);
            put_entity(e, &main_rq);
        }
        rq_unlock(&main_rq);
        rq_unlock(&nosched_rq);
        delay(1);
    }
}
start() {
        thread_create(NULL, NULL, gang_sched, NULL);
}
```

Results

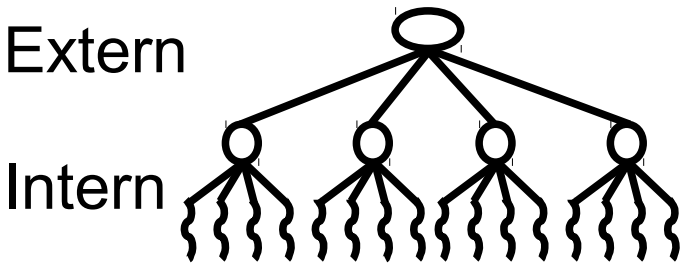

Bubble-gang
Marcel-shared
Linux
Speedup
Number of jobs
0
1
2
3
4
1
2
3
4
5

Space-Irregular parallelism BT-MZ

- OpenMP Multi-Zone version of NPB (NASA Parallel Benchmark): BT-MZ

- Two-level parallelism:
  - External 16 irregular (x,y) regions
  - Internal z regular parallelism

- Load estimation is known



A “spread” scheduler

Kind of bin packing

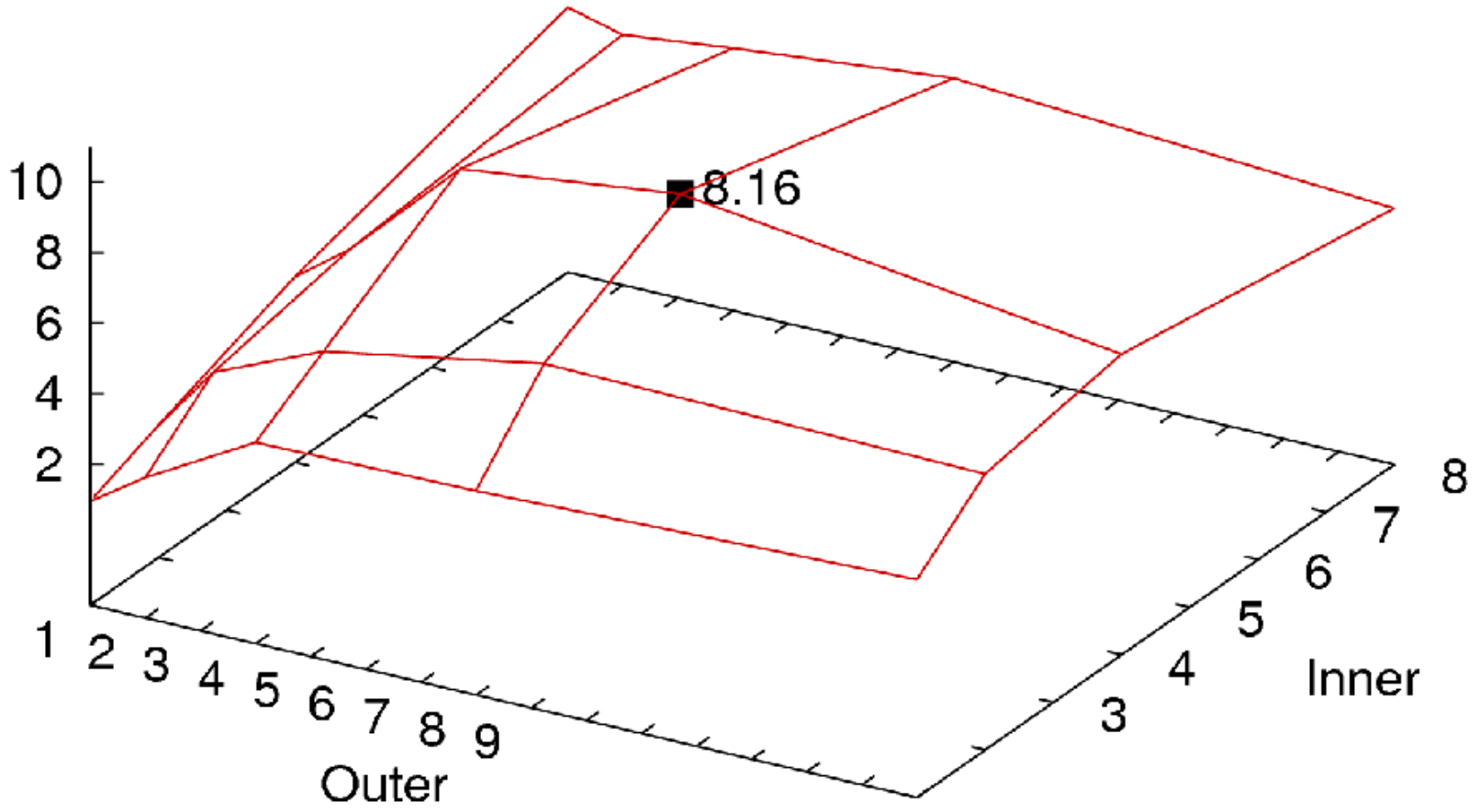
- Sort threads and bubbles by application-provided computation load estimation
- “Explode” very big bubbles
- Greedily distribute on top-level runqueues
- Recurse into sub-runqueues

- May take into account resource usage
  - Memory, bandwidth, ...

Spread scheduler in action

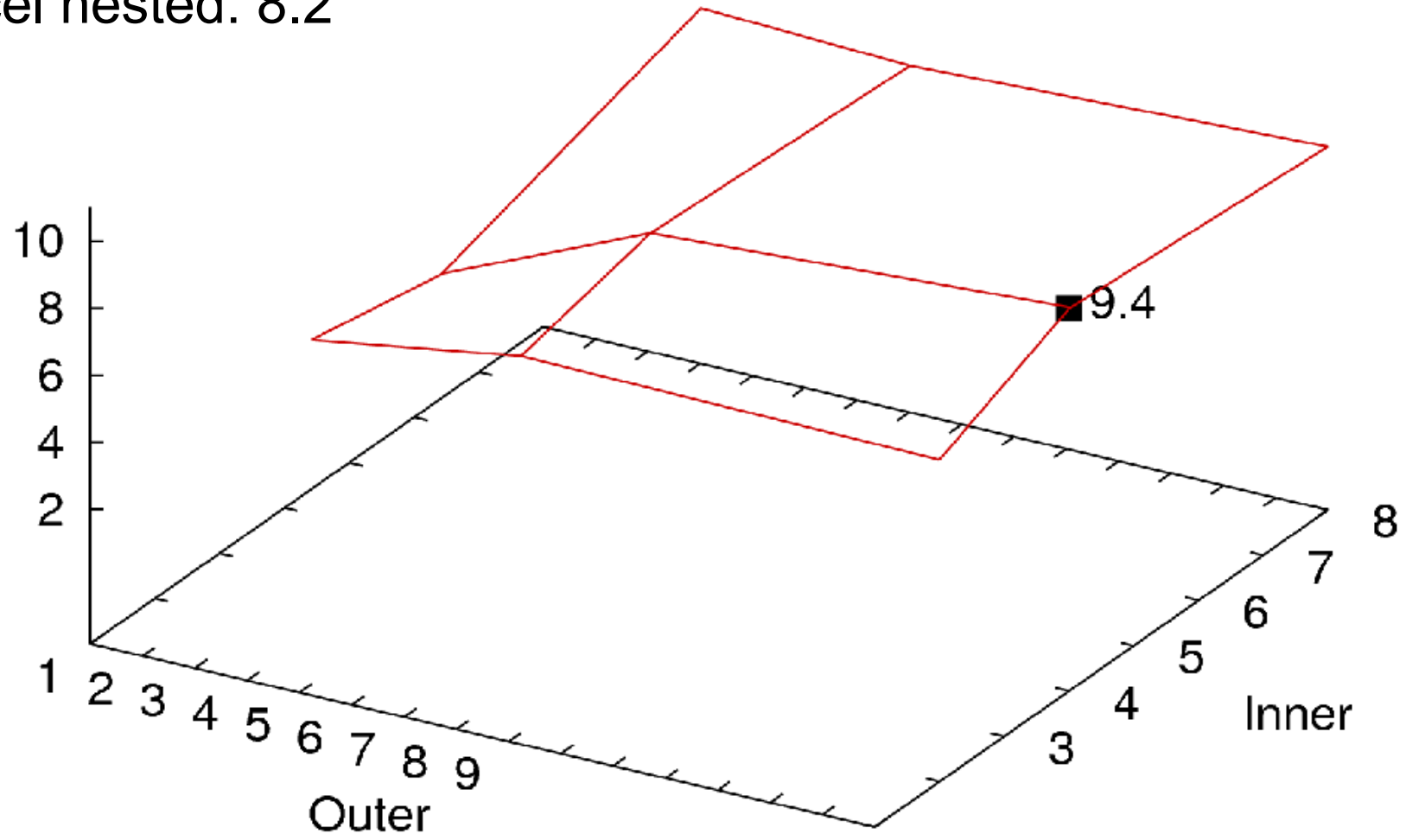
Nested Parallelism Marcel

NPTL nested: 6.3



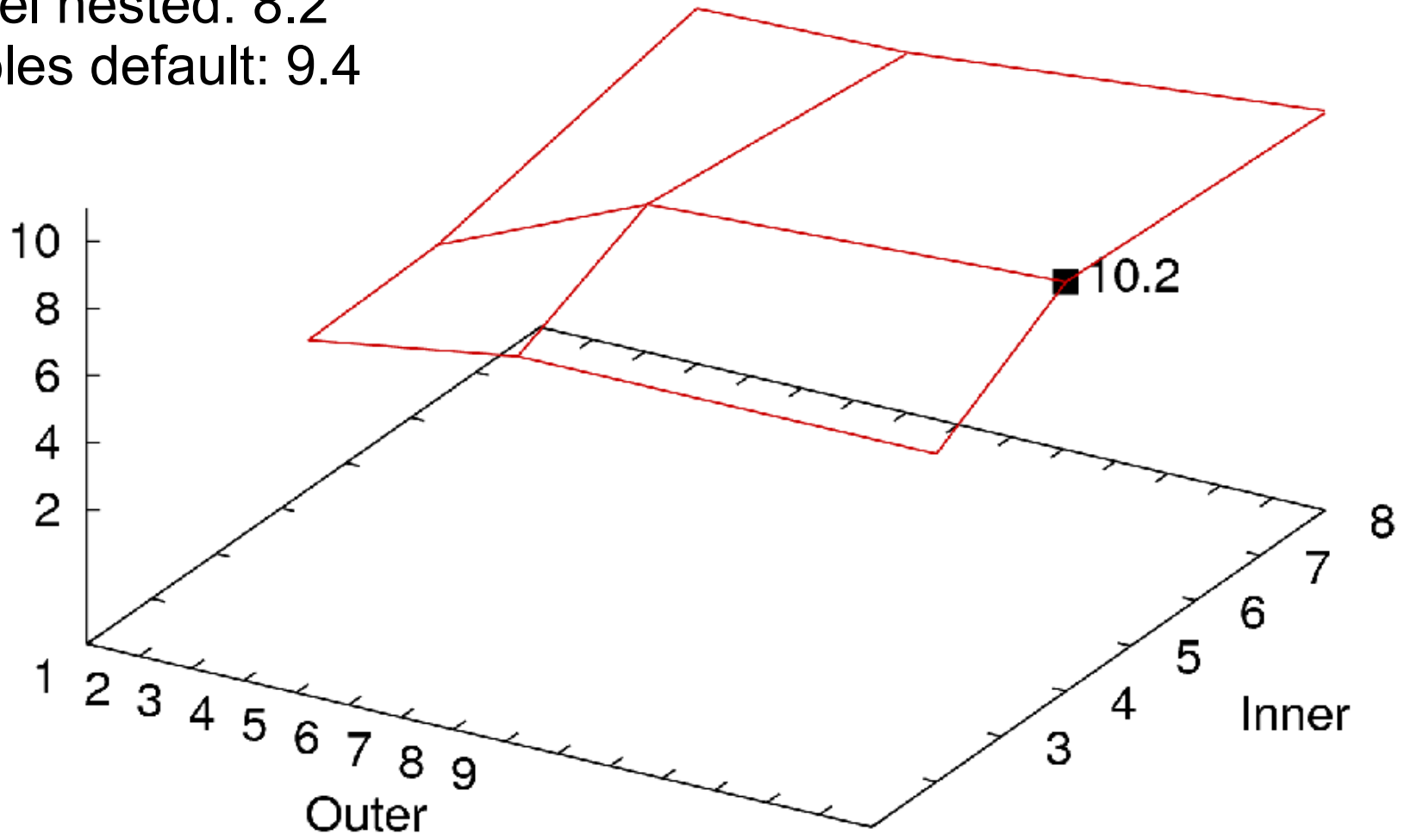
Nested Parallelism Marcel with Bubbles

NPTL nested: 6.3
Marcel nested: 8.2



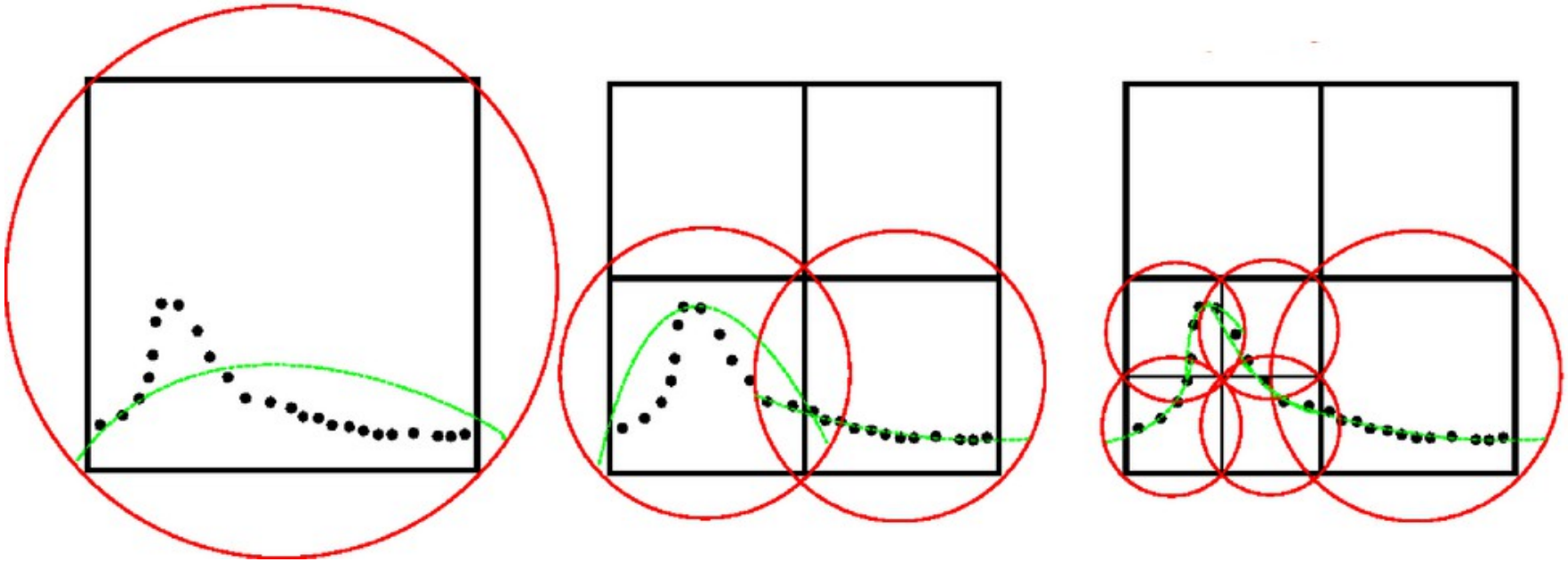
Nested Parallelism Marcel with tuned Bubbles

NPTL nested: 6.3
Marcel nested: 8.2
Bubbles default: 9.4



Time-irregular parallelism MPU

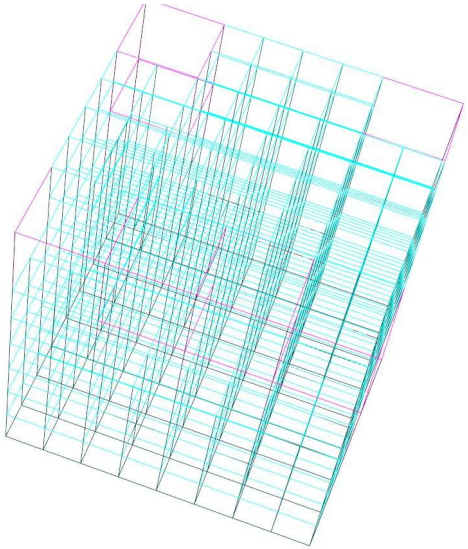
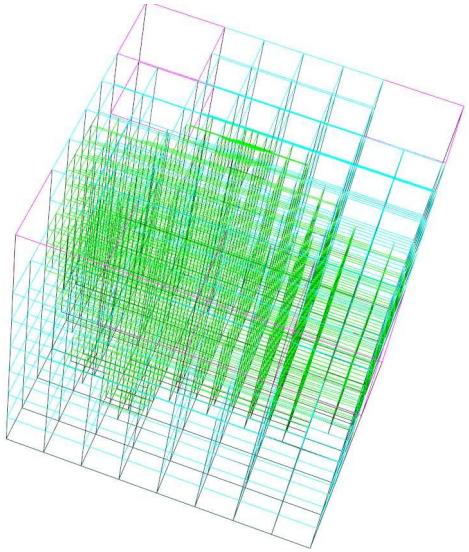
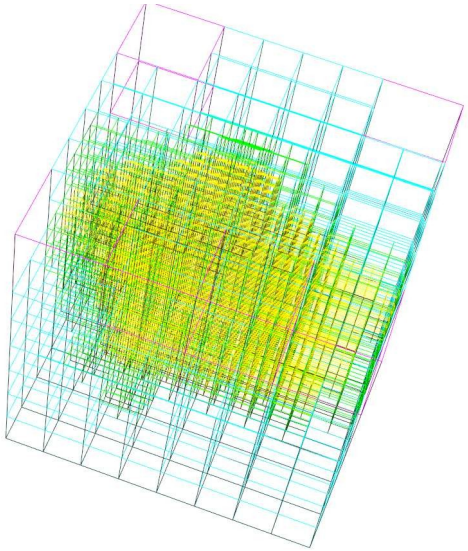
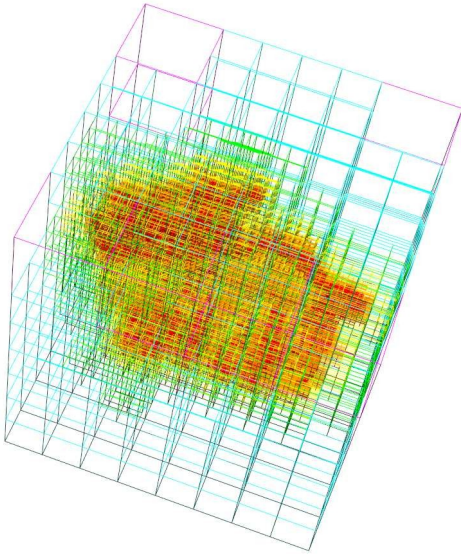
- Surface reconstruction from point set
  - Schlick, Boubekeur, Diakhate
- Irregularly recursively parallel
  - Don't know where to refine *a priori*

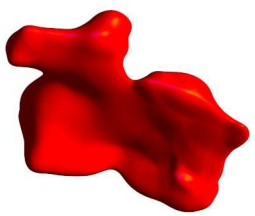
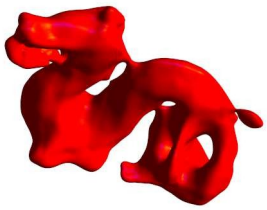
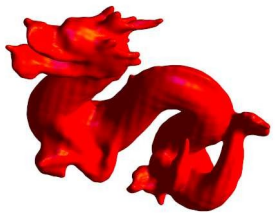
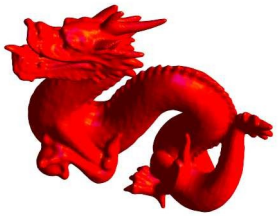


Parallelization of MPU

```
void Node::compute(){

  computeApprox();

  if(_error > _max_error){
    splitCell();

    #pragma omp parallel for
    for(int i=0; i<8; i++)
      _children[i]->compute();
  }
}
```

Time-irregular parallelism MPU

How to schedule MPU threads?

- A very large number of threads
- Irregular creation
- No load estimation

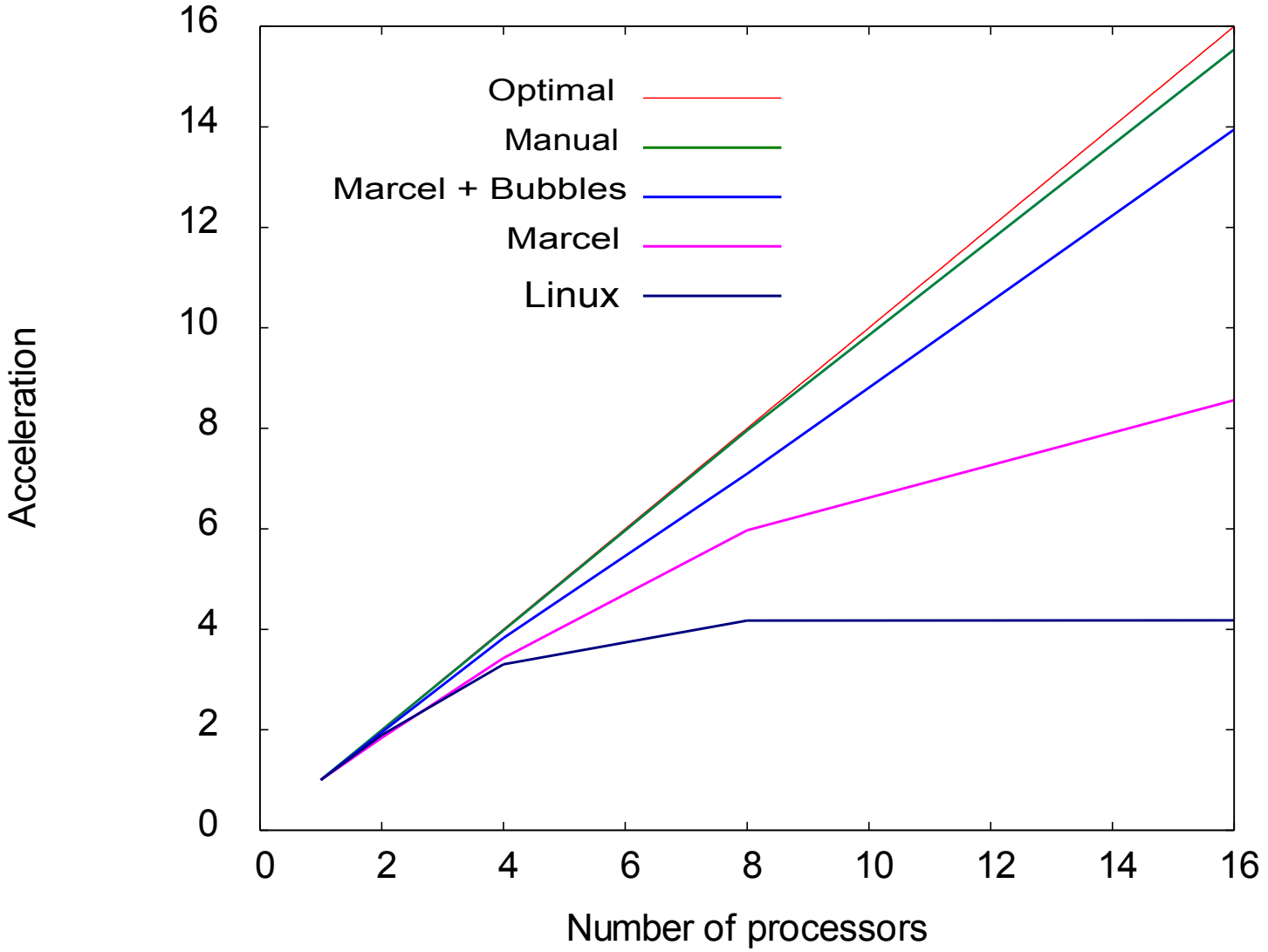
➔ Requires dynamic load balancing

Affinity scheduler

- Affinity-bound distribution
  - Broquedis
- Affinity and memory -aware stealing
  - Steal locally
  - Tear bubbles
  - Migrate memory
    - Jeuland

✔ Suited to dynamic nested parallelism

MPU Results


Optimal
Manual
Marcel + Bubbles
Marcel
Linux
Acceleration
Number of processors
0
2
4
6
8
10
12
14
16

Conclusion
A new scheduling approach

Structure & conquer!

- Bubbles = simple yet powerful abstractions
  - Recursive decomposition schemes
    - Divide & Conquer
    - OpenMP
- Implement scheduling strategies for hierarchical machines
  - A lot of technical work is saved
- Significant benefits
  - 20-40%

Conclusion
A new scheduling approach

- Experimentation platform
  - Available for various high-end machines
    - Bull Fame2, SGI Altix, AMD Opteron...
  - Quickly test existing strategies or combinations
    - With action replay feedback
- Visibility
  - Used in ANR CIGC projects: PARA, NUMASIS
  - OpenMP backend
  - http://pm2.gforge.inria.fr/

Future work

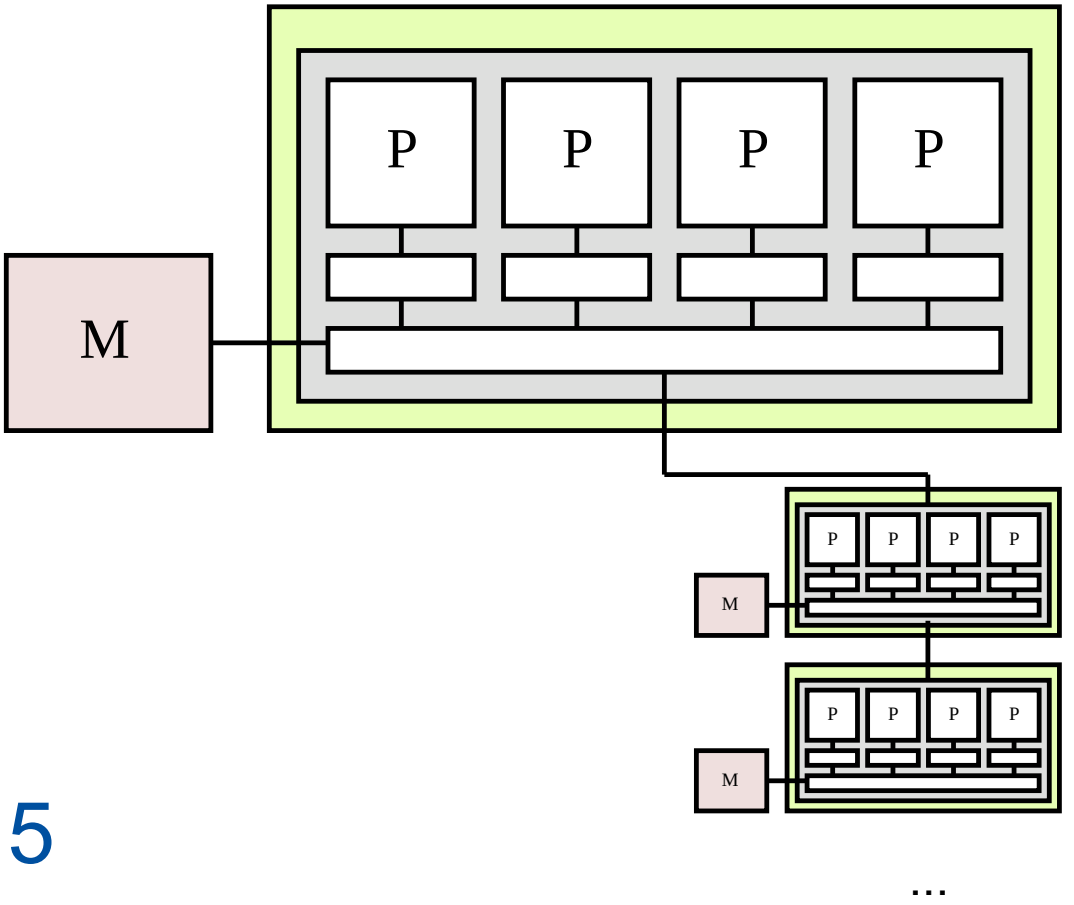
- Information from programmer and/or compiler, e.g. OpenMP
  - Thread-data relationships
  - Big application steps
    - Initialization, loop boundaries
  - François Broquedis PhD thesis
- More abstract languages for bubbles
  - ML, Prolog ?

Future work

- Spread the idea
  - Other OpenMP compilers
  - Linux
  - Xen
- Next architecture targets
  - 2D meshes
  - Heterogeneity (*PUs, NCC-NUMA)
  - Sea of cores
    - Embarrassingly parallel machines

Supplement

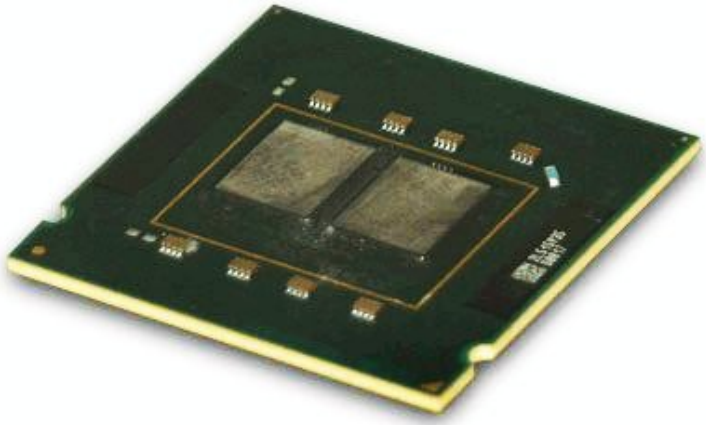
AMD Quad-Core

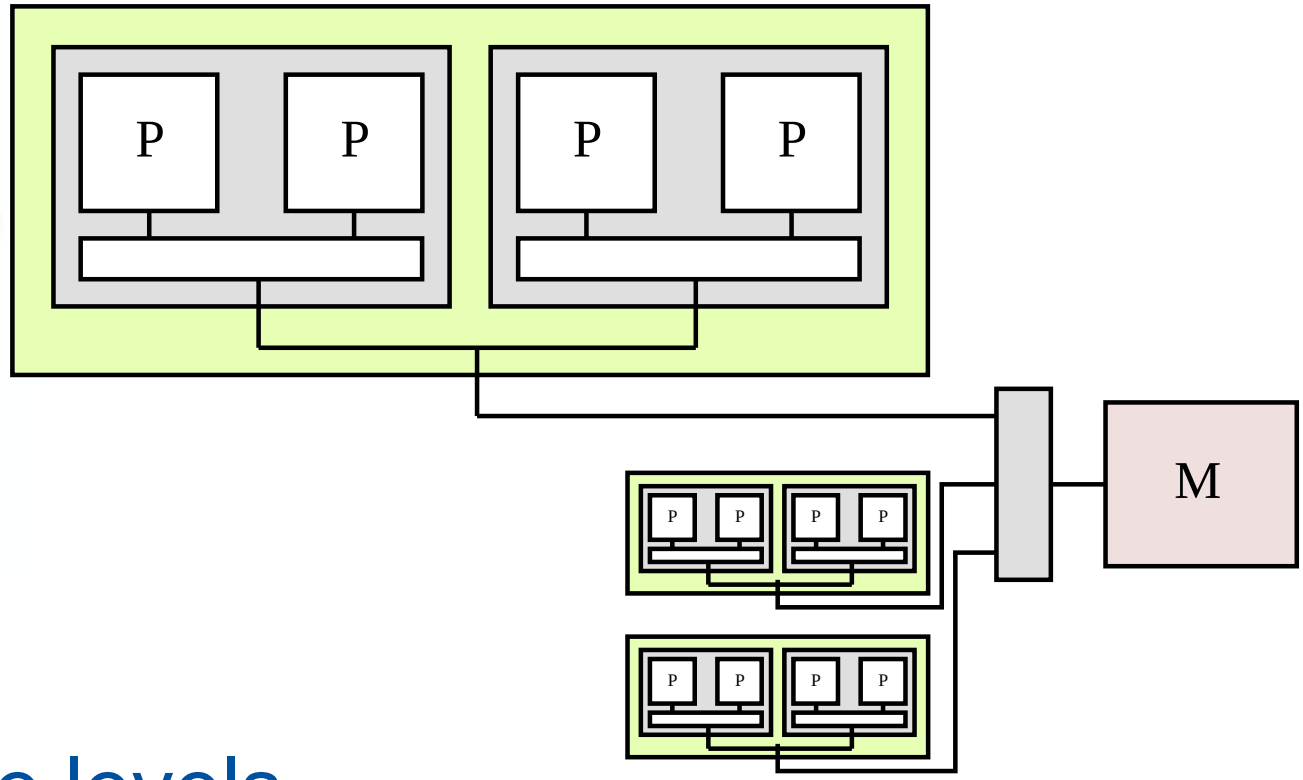


Shared L3 cache

NUMA factor ~1.1-1.5

Intel Quad-Core





Hierarchical cache levels

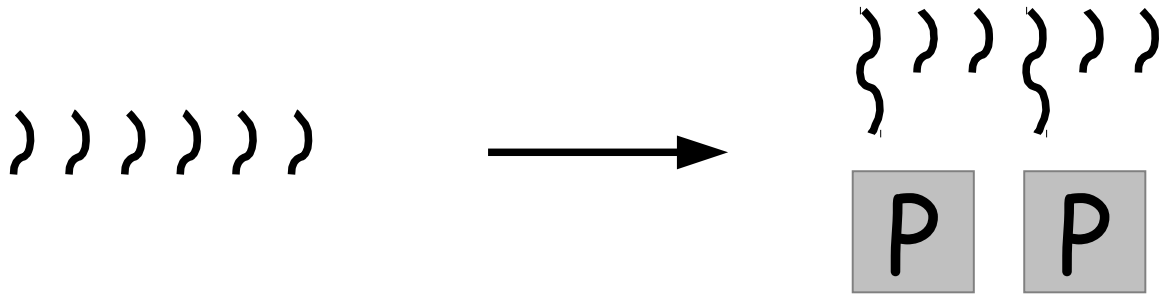
Work stealing

```
idle() {
  look_up(self_rq);
}

look_up(rq) {
  if (look_down(rq->father, rq))
    return;
  look_up(rq->rather);
}

look_down(rq, me) {
  if (look(rq))
    return;
  for (i=0; i<rq->arity; i++)
    if (rq->sons[i] != me)
      look_down(rq->sons[i], rq);
}
```

```
look(rq) {
  b = find_interesting_
        bubble(rq)));
  if (!b)
    return 0;
  rq_lock(rq);
  get_entity(b);
  rq_unlock(rq);
  rq_lock(self_rq);
  put_entity(b, self_rq);
  rq_unlock(self_rq);
  return 1;
}
```

Thread seeds

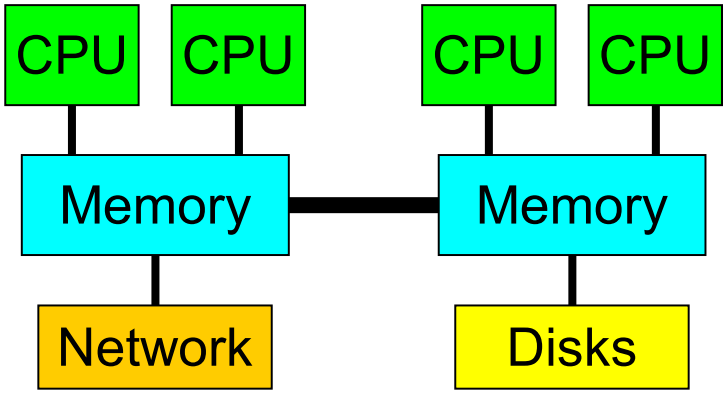
- Lazy creation of threads
  - Makes creation of a team of threads lightweight
  - Fast sequential execution on a single processor
  - Suited parallel execution on several processors

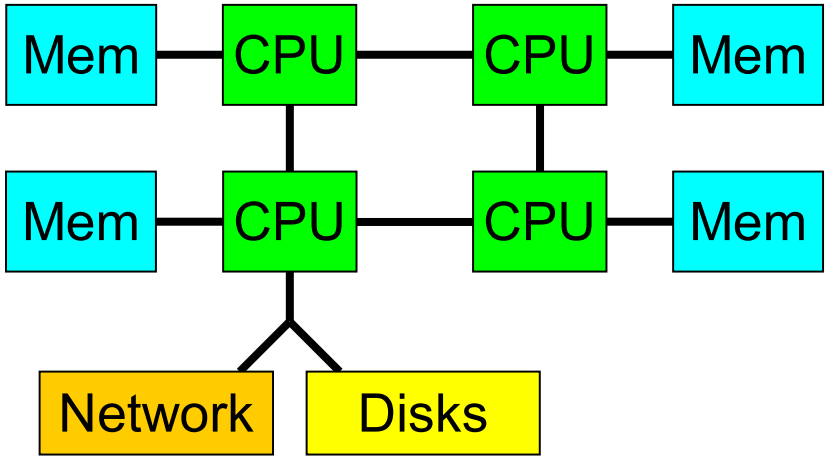


- Well suited to irregular nested OpenMP parallel sections

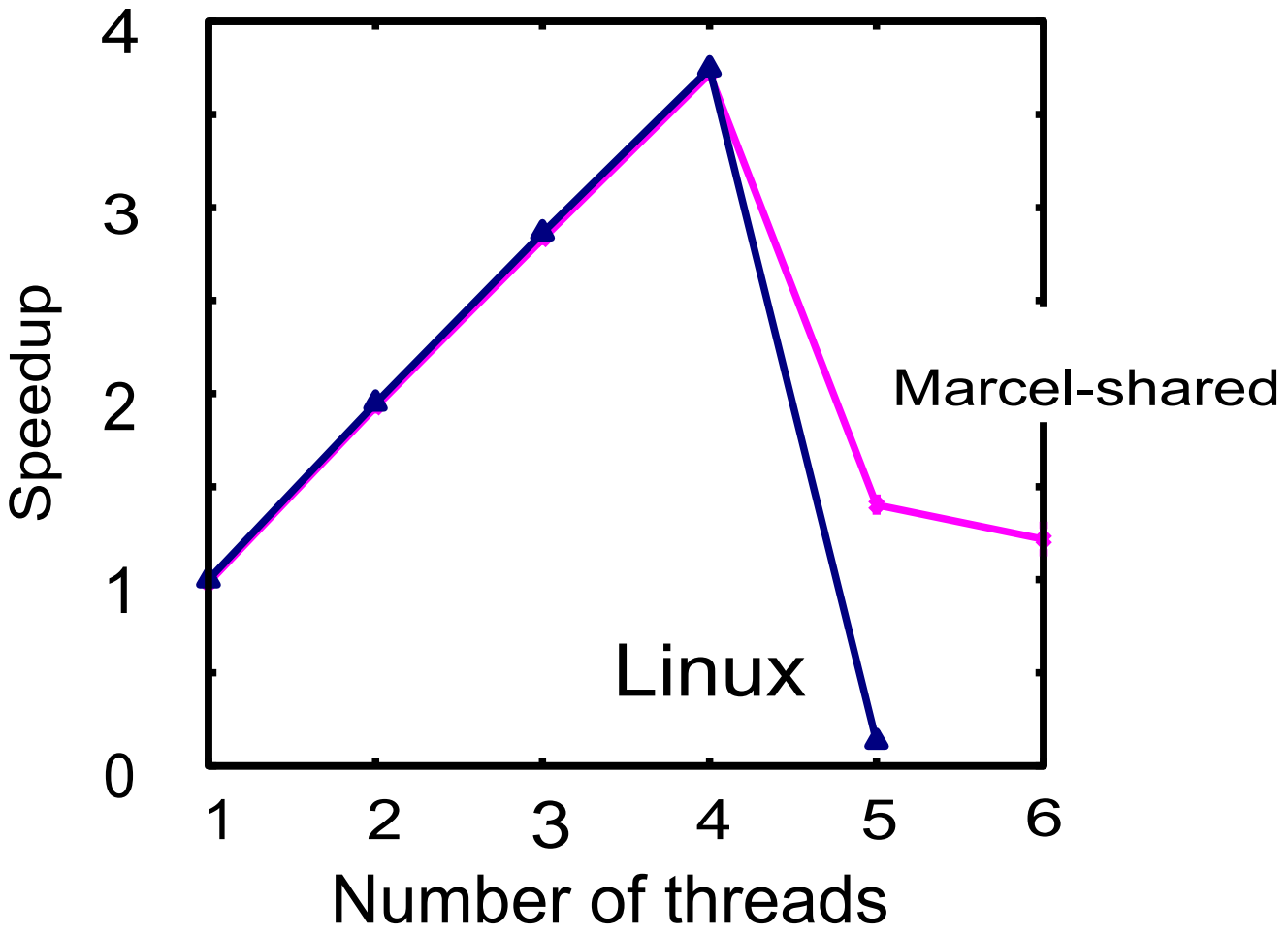
Taking NUMA effects into account during I/O

- Peripherals affinity
  - Located on an I/O bus linked with a memory bus
    - Closer to some processors and memory banks



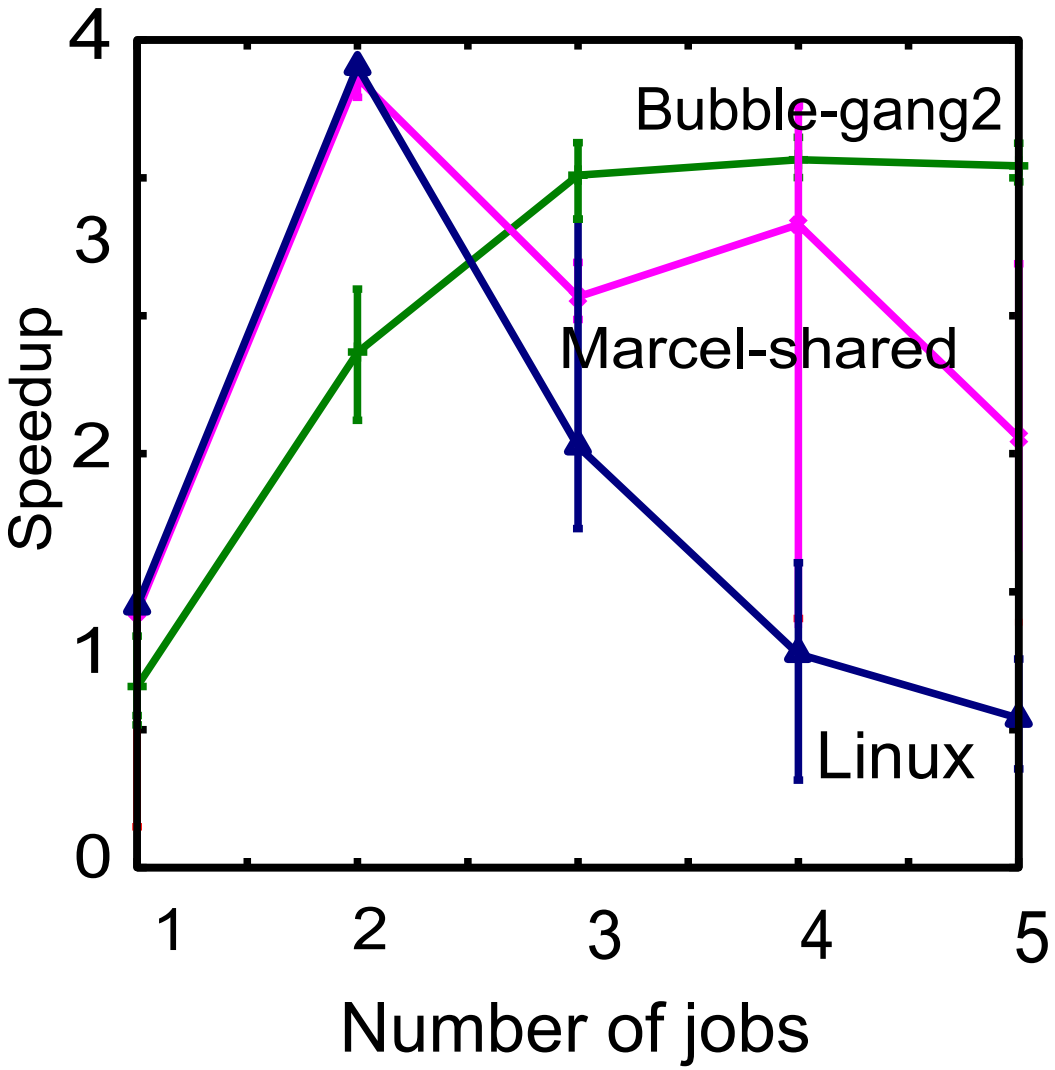


Parallelization of one job



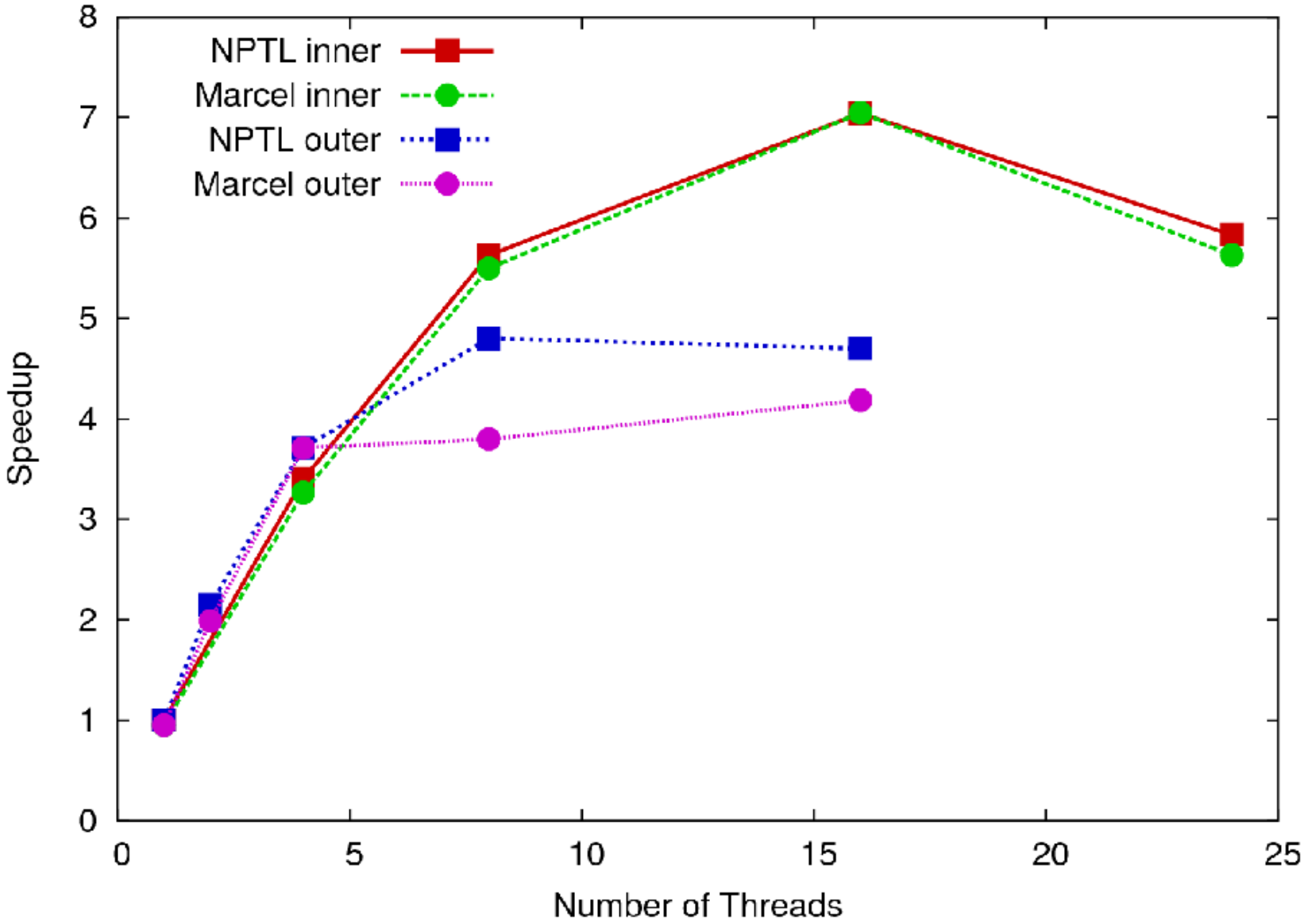
Performant as long as no overloading is used

2-way parallelization?



Gang Scheduling

Non-nested Parallelism



Nested Parallelism NPTL (Linux 2.6.17)

NPTL inner: 7.0
NPTL outer: 4.8