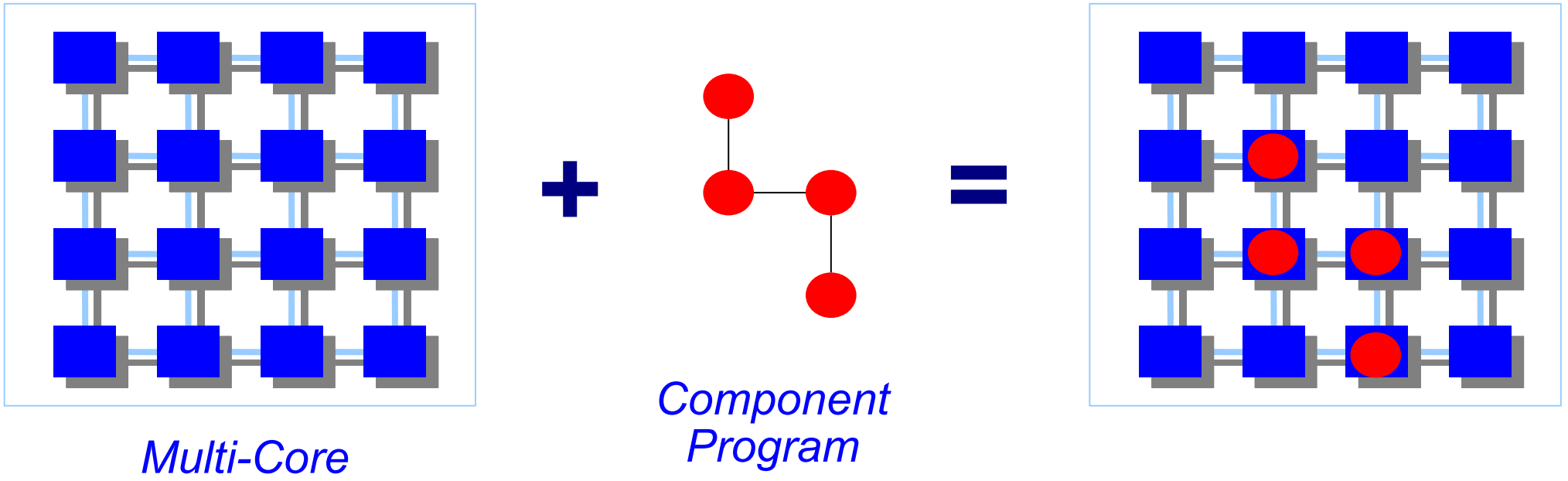
CAPSULE: Parallel Execution of Component-Based Programs

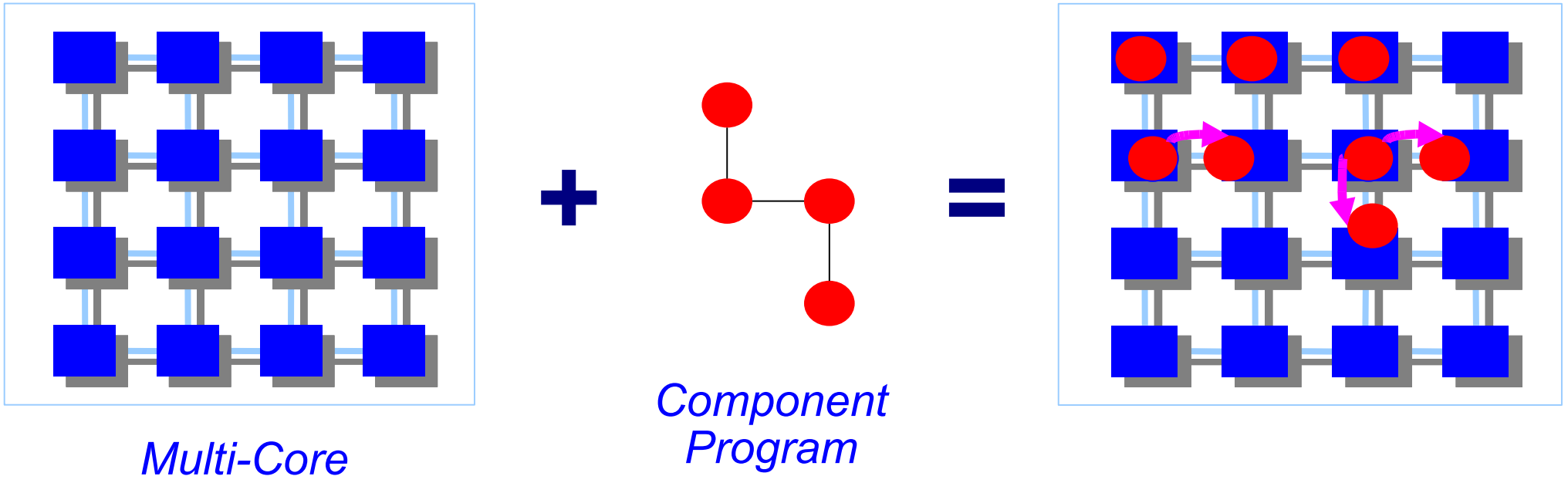
Pierre Palatin, Yves Lhuillier, Olivier Temam
INRIA, France

Software & Hardware Trends Blend Well



- Software components:
  - Separate/Encapsulated software parts
  - Explicit communication links

Component Programming Is Not Enough



- **BUT** component programming ≠ finding parallelism
- Augment components with ability to **divide**
  - BUT locks still managed by user
- Granularity & Mapping too complex to be exposed to the user

  ⇒ **architecture or run-time system support**
- User: where division may occur
- Architecture/Run-Time System: when division does occur

Probing & Division

Normal loop

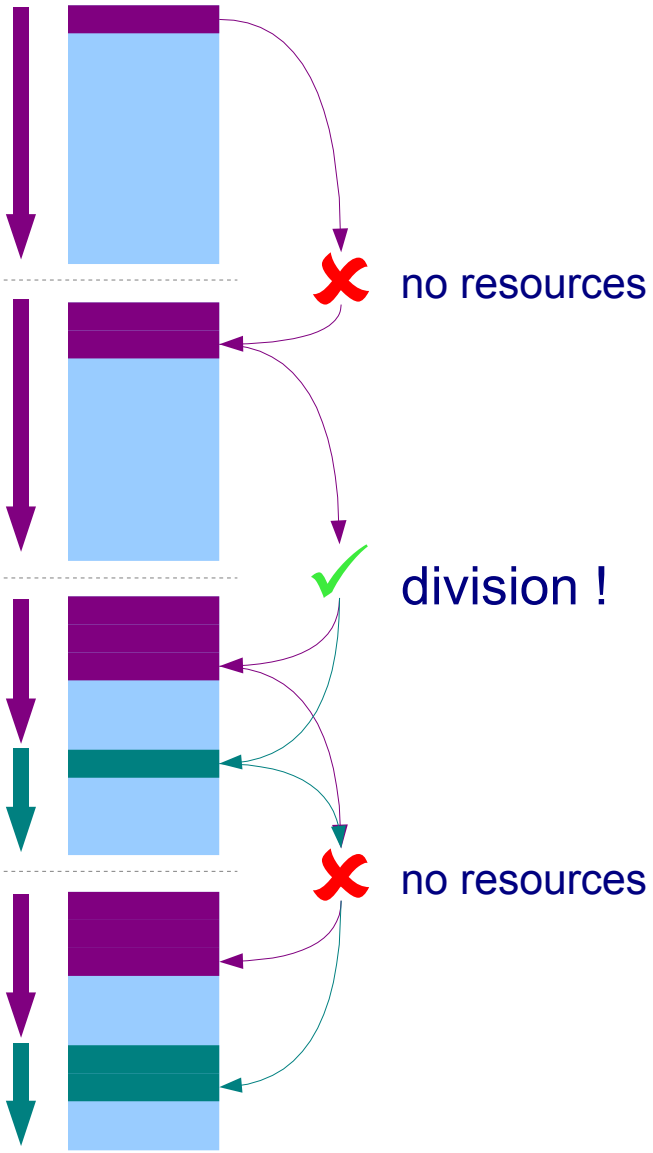
```
idx = min;
while (idx < max) {
        c[idx] = a[idx] + b[idx];
        idx++;
```

Division code

```
        ctxt = capsys_probe(&loop);
        if (ctxt) {
                newmin = (idx+max) / 2;
                newmax = max;
                max = newmin – 1;
                … //Prepare “arg”
                capsys_divide(ctxt, arg);
        }
}
```

Split bounds

Low-Level C Implementation

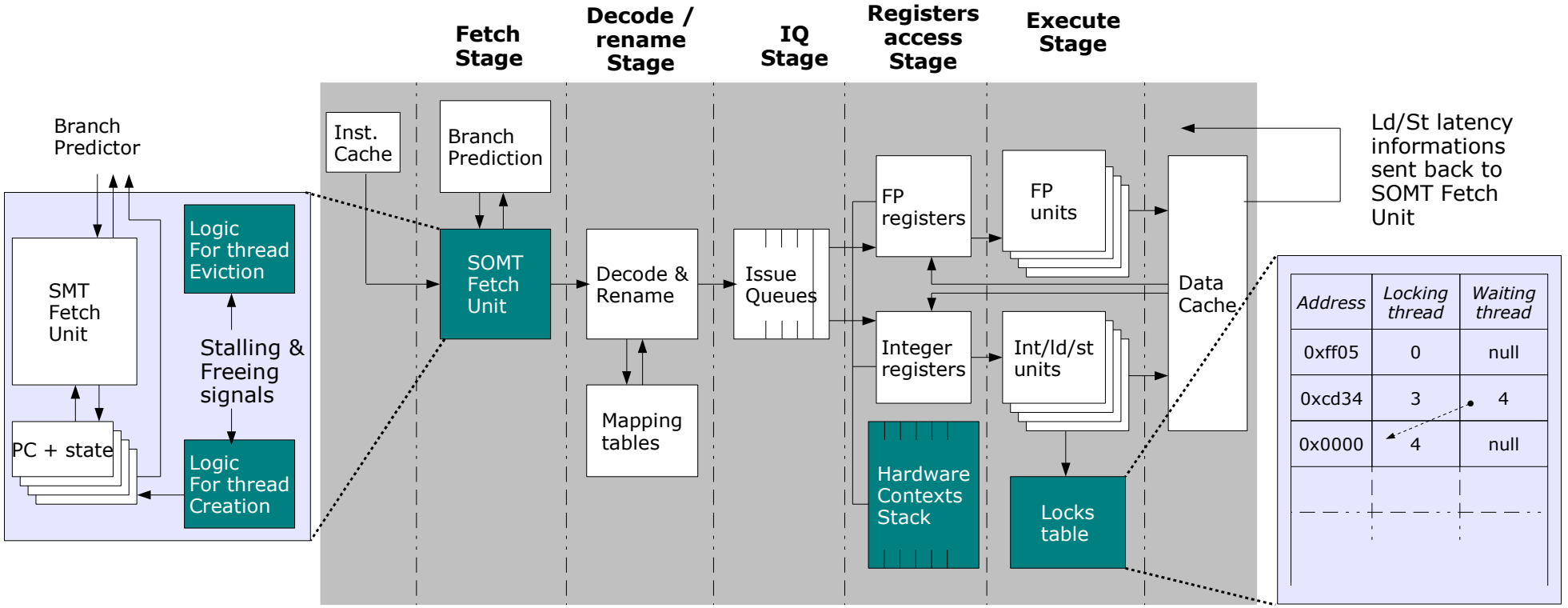


“do loop”

- **Probing & division** must be very efficient

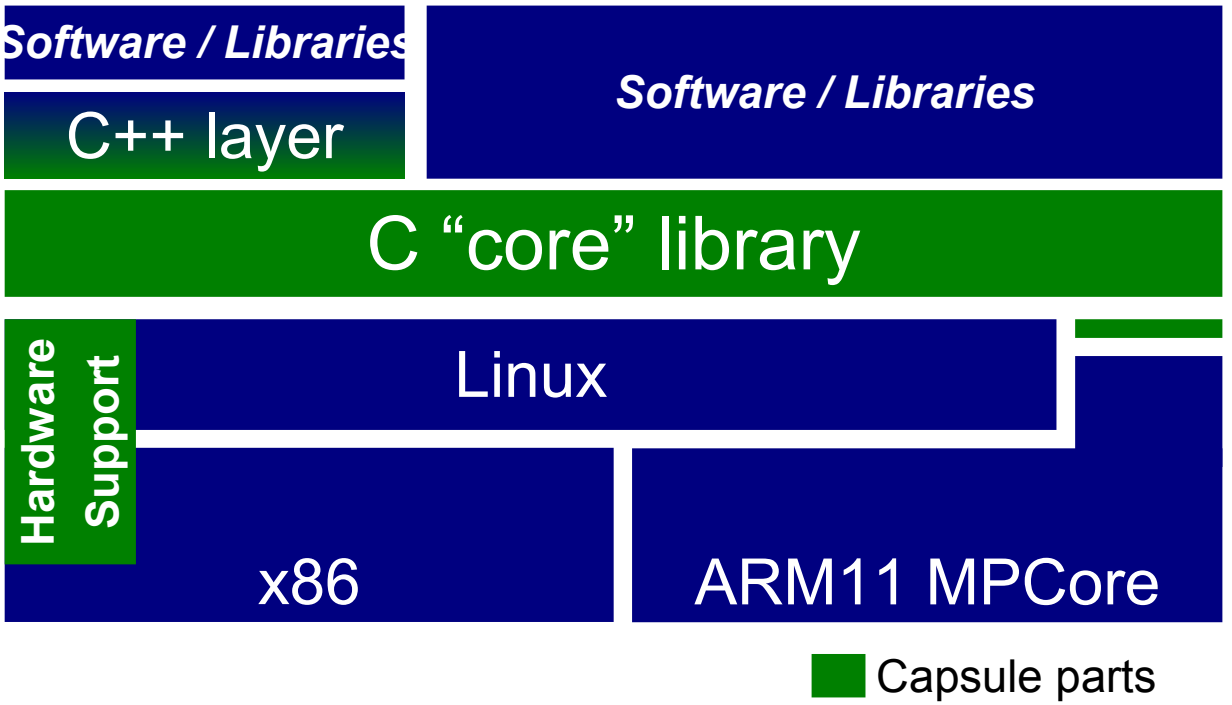
With Hardware Support

- Target architecture: SMT (8 threads)
- 1 component ~ 1 thread
- Dynamic thread creation (`nthr`)
- Division steering



P. Palatin, Y. Lhuillier, O. Temam: Capsule, Hardware-Assisted Parallel Execution of Component-Based Programs, MICRO 2006

Software-Only Version

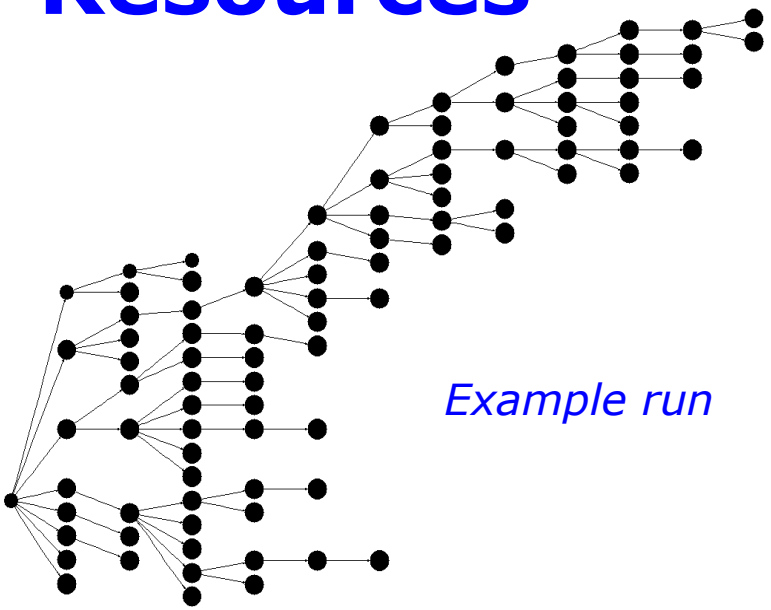


- Low-Level C-layer for speed and portability; back-end for other // frameworks (OpenMP, STAPL, ...)
- Higher-Level library in development
- Extends threading layer (pthread for now)
- Shared-memory CMPs (distributed-memory CMPs in the works)
- Probing ~ 10 cycles, division ~ 1400 cycles Intel on Core Duo

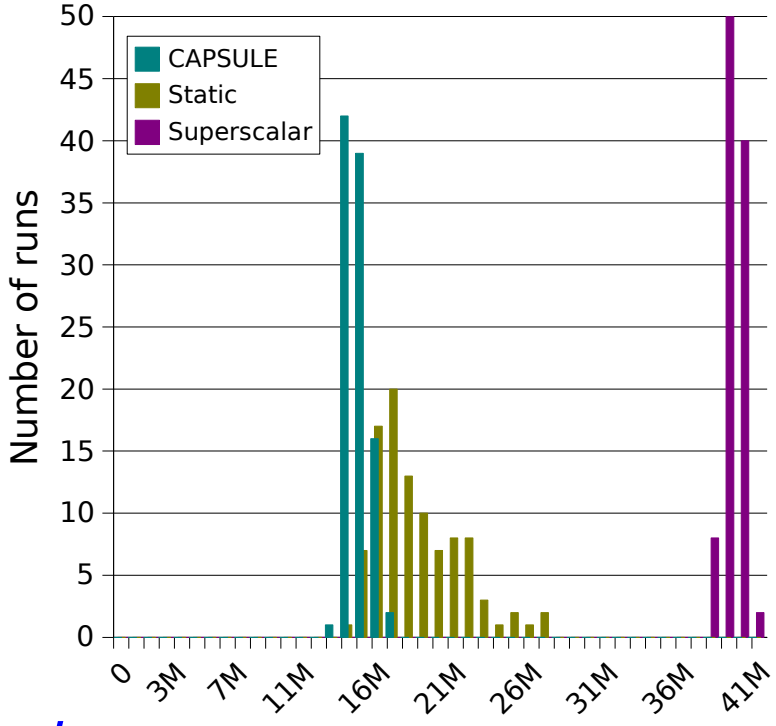
https://alchemy.futurs.inria.fr/capsule

Dynamic Exploitation of Resources

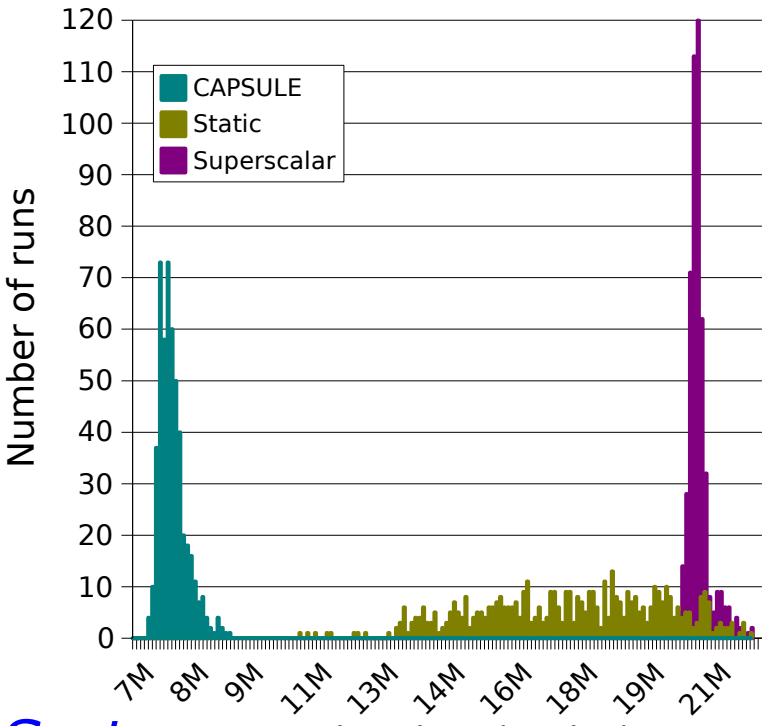
- Load-Balancing properties vs. static mapping
- Performance stability across data sets
  - real-time systems



Example run



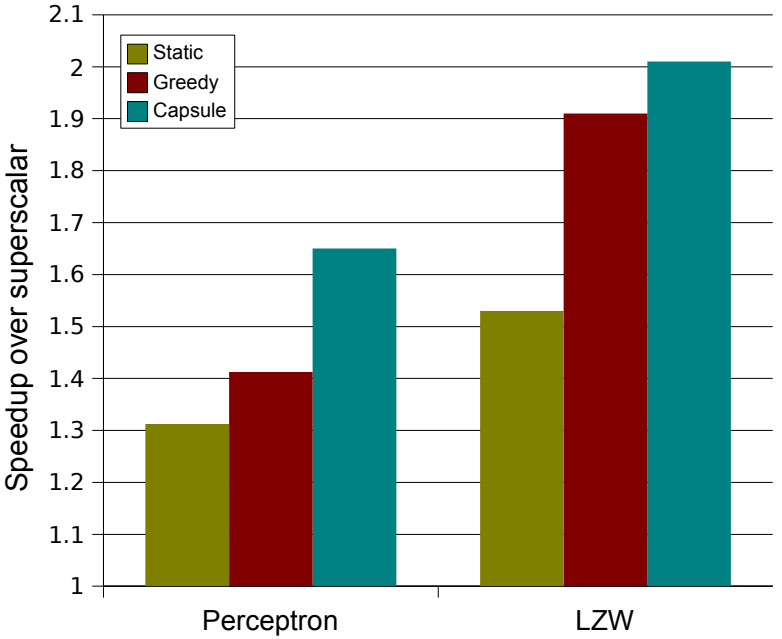
Dijkstra

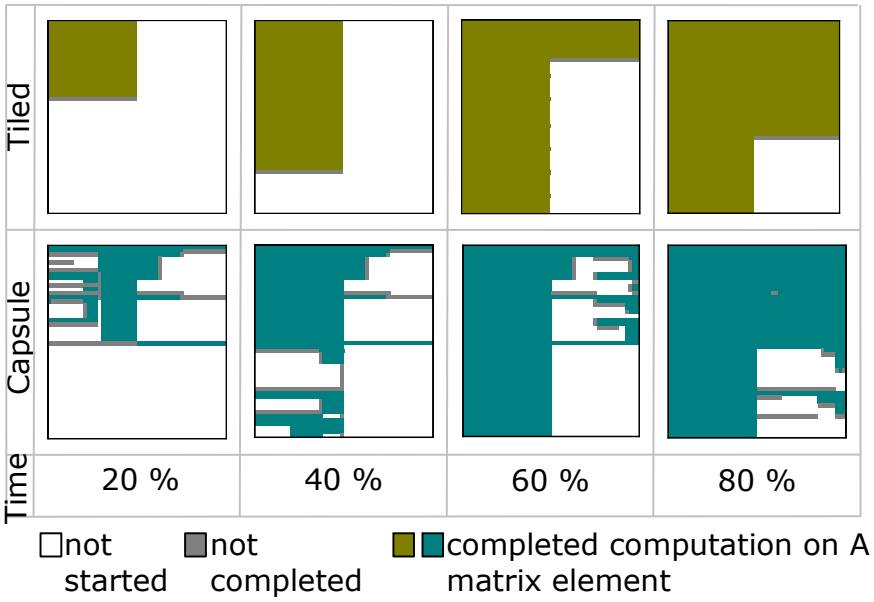


QuickSort

Thread Steering & Swapping

- Overhead issue of too fine-grain parallelization
  - Greedy = default division policy
  - Safeguard: slow down division if threads termination rate too high
- Thread swapping
  - Stack of stalled threads
  - Swap out threads with long latency loads
  - Classic, BUT, combined with task division:
    - breaks execution into chunks (~blocks of iterations)
    - can emulate some static locality optimizations

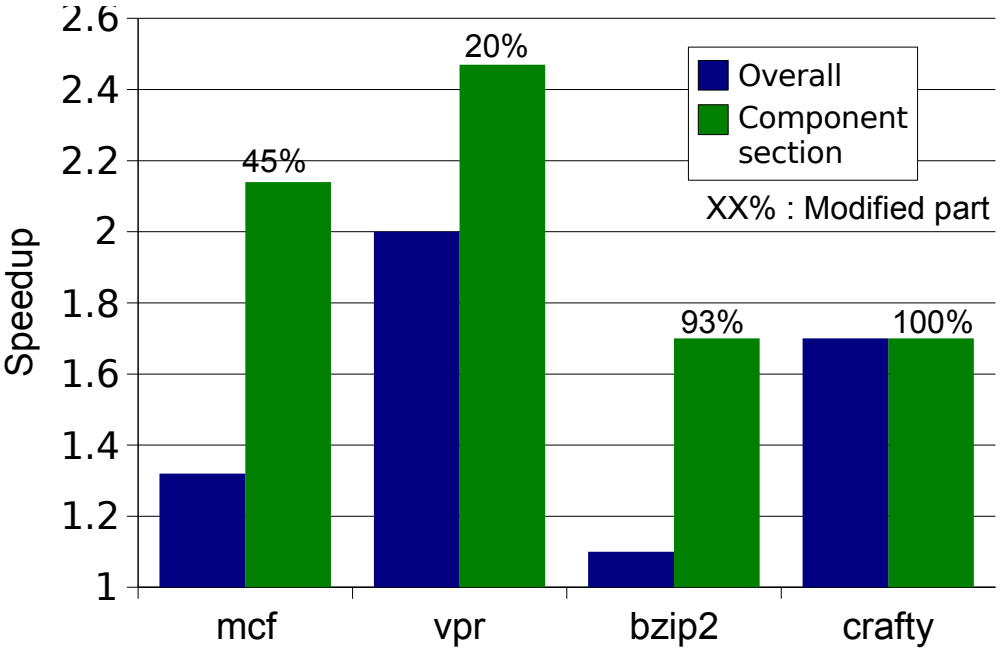




Matrix-Vector Multiply

Larger Applications (SPEC-CINT2000)

- Partly reverse-engineered and reprogrammed
- Moderate code modifications BUT
  - component decomposition easier when starting from program specification
  - better suited to new implementations rather than existing programs
- HOWEVER components facilitate/favor program reuse



Speedup over original SPEC benchmarks

Benchmark	Lines	Modified or added functions	Modified or added lines	% total execution time
mcf	2412	2	174	45%
vpr	17729	10	624	93%
bzip2	4649	3	317	20%
crafty	45000	8	201	100%

Modifications for componentization

Benchmark	# divisions requested	# divisions granted	% divisions granted	#insts per division granted
mcf	99598	40532	40%	3.7K
vpr	67560	2702	4%	4.5M
bzip2	38656	2319	6%	30M

Percentage and rate of successful divisions

- Few probes result into actual divisions
- Low overhead of probes

Conclusions & Future Work

- Main contributions
  - Relieve user of parallelization granularity & mapping issues
  - Compatible/Designed for “complex” applications
  - Leverage an existing trend in software engineering
  - Tight program/hardware (or run-time) integration
- Future work
  - Application to distributed-memory CMPs w/ & w/o hardware support
  - High-Level library for a more simple syntax
  - Disseminate through a few popular applications (x264, Gzip,...)
  - Real-Time & QoS

Thank You !