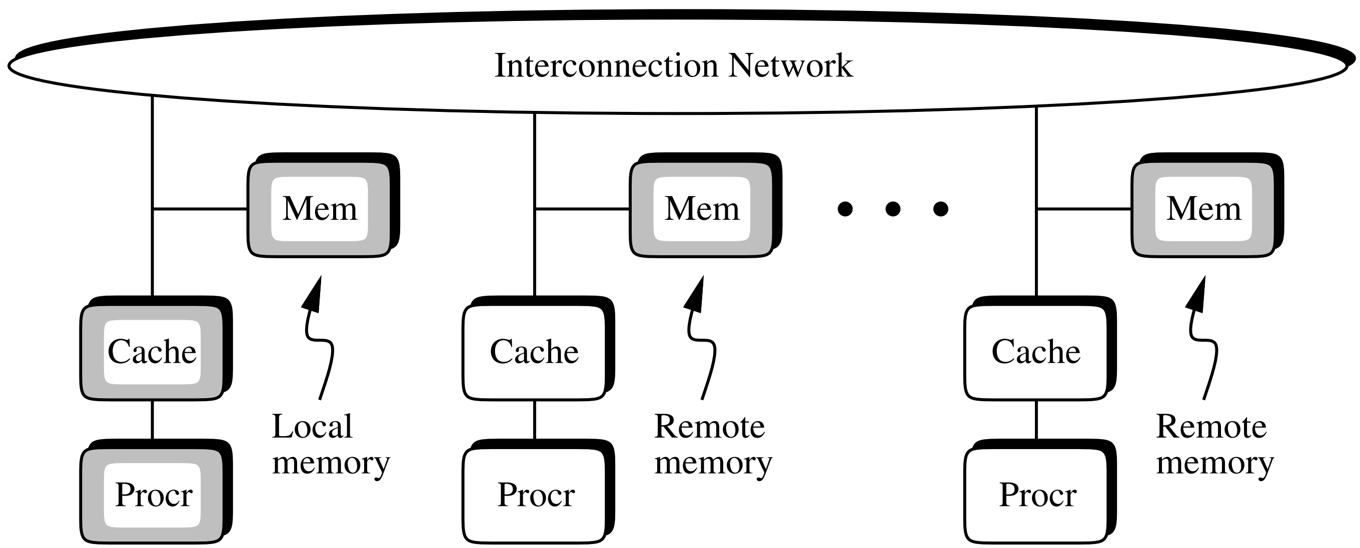
Array Data Layout for the Reduction of Cache Conflicts

Naraig Manjikian and Tarek S. Abdelrahman

Department of Electrical and Computer Engineering
University of Toronto, Ontario, Canada

21 September 1995

Scalable Shared-memory Multiprocessors



- Production systems: Convex SPP, KSR1/2, Cray T3D
- Research systems: DASH/FLASH, NUMAchine

Locality Enhancement in Data-parallel Programs

```
doall j = ...
  doall i = ...
     a[i,j] = ...
  end do
end do
   :
doall j = ...
  doall i = ...
     ... = a[i,j-1] + a[i-1,j] + ...
  end do
end do
```

- Identify parallelism *and* data reuse
- Performance maximized by exploiting parallelism *and* enhancing locality

Locality Enhancement for Parallel Loops

- Transformations to exploit reuse and retain data in cache:
  - *tiling/blocking* for individual loop nests
  - *fusion* for multiple loop nests

- Benefit of locality enhancment is diminished by *cache conflicts*:
  - eject reusable data from the cache
  - difficult to predict (factors: hardware, array sizes, access patterns)

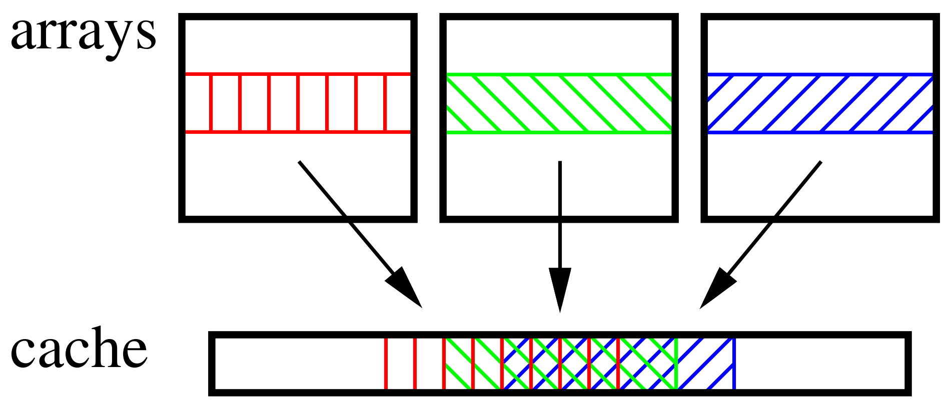
- <u>*Our contribution:*</u> cache partitioning to eliminate cache conflicts
  - adjust memory layout to control mapping in cache
  - maximize benefit of locality-enhancing transformations
  - experimental results demonstrate utility/effectiveness

Outline

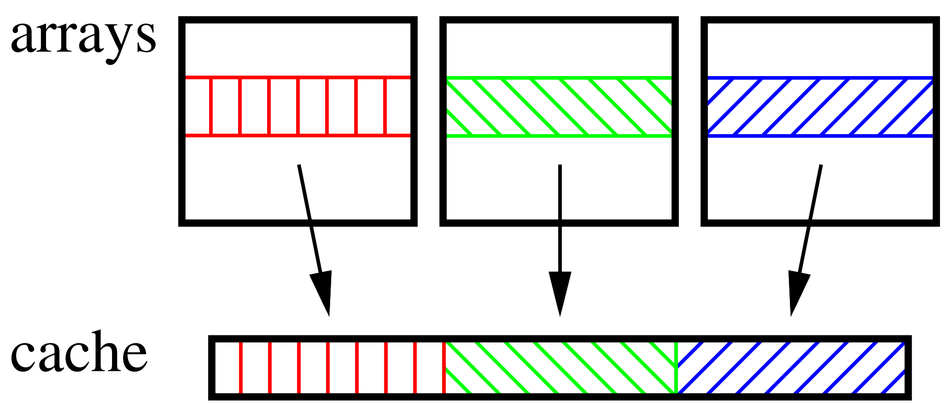
1. Cache partitioning
2. Benefits
3. Experimental results
4. Conclusions

Cache Partitioning to Eliminate Conflicts

- Cache conflicts occur when data mapping overlaps in cache:



- Our solution: adjust memory layout to prevent data from overlapping:

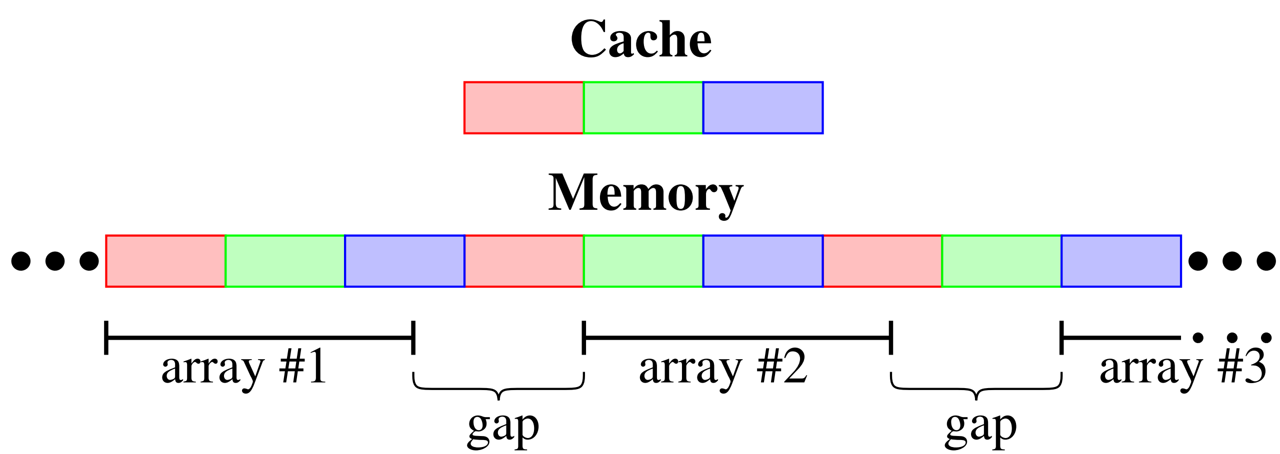


Cache Partitioning to Eliminate Conflicts (cont'd)

- Assumption: data accesses are compatible

Compatible data accesses   Incompatible data accesses

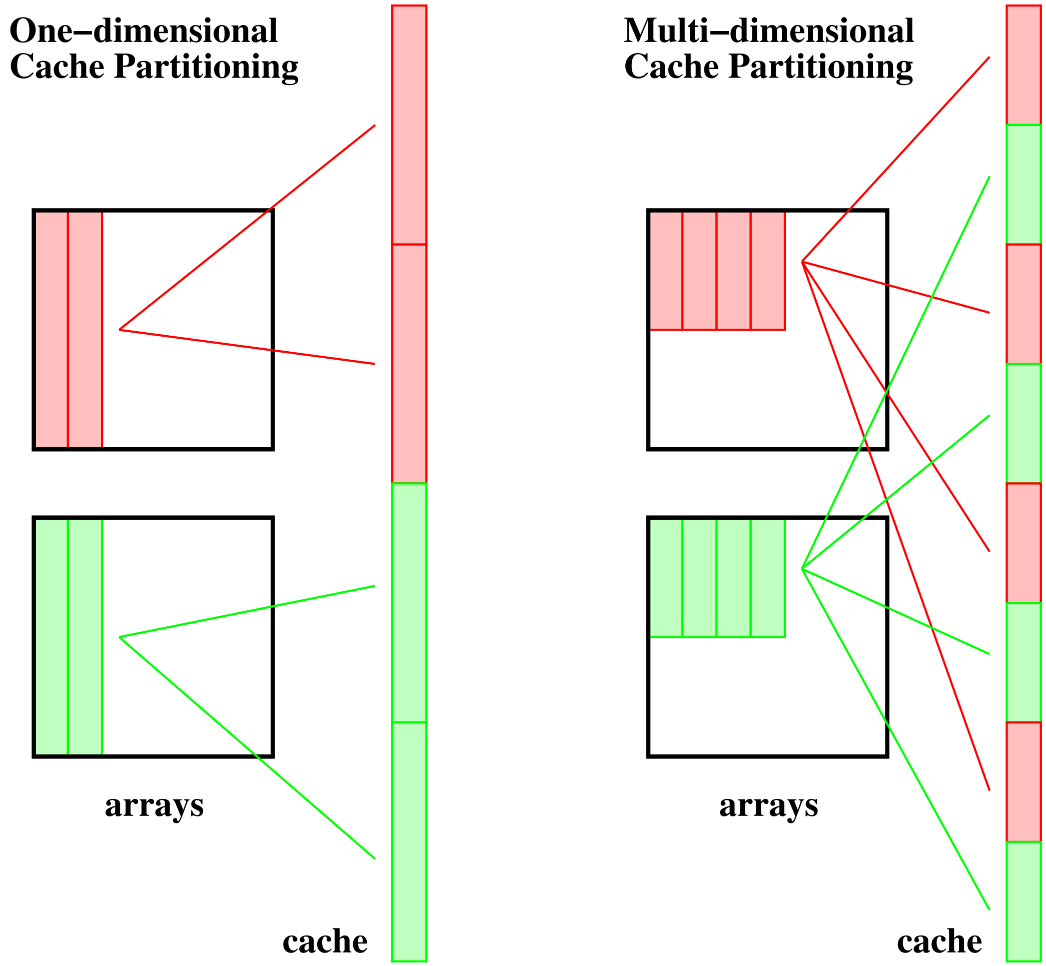
- Adjust starting addresses with gaps to map arrays to different partitions:



```
GreedyMemoryLayout (A,c):: // A = arrays, c = cache size
    n_a = |A|
    s_p = c / n_a
    P = {0, 1, ..., n_a - 1}
    q = q_0
    do
        select a ∈ A
        mapped_address = CacheMap (q)
        foreach p ∈ P do
            target_address(p) = p · s_p
            gap(p) = target_address(p) - mapped_address
            if target_address(p) < mapped_address then
                gap(p) = gap(p) + c
            endif
        endfor
        select p_opt ∈ P where gap(p_opt) = min_{p∈P} gap(p)
        P = P \ {p_opt}
        START(a) = q + gap(p_opt)
        q = START(a) + SIZE(a)
        A = A \ {a}
    while A ≠ ∅
```

Cache Partitioning to Eliminate Conflicts (cont'd)

- Cache partitioning may be performed on more that one dimension:

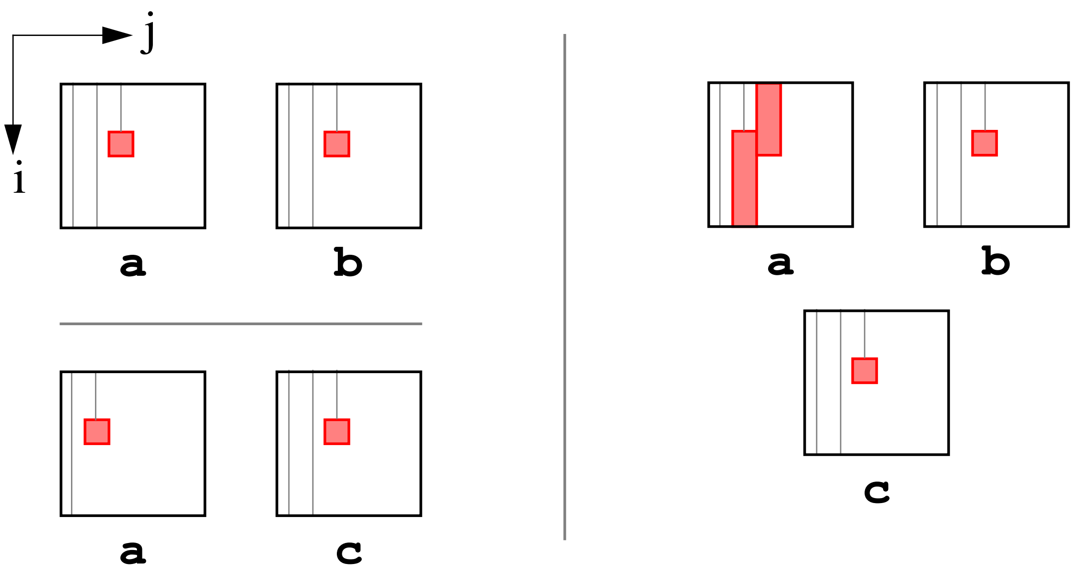


Benefits of Cache Partitioning

- *Fusion* of loops reusing arrays:

```
do j=2,n-1
  do i=2,n-1
    b(i,j)=a(i,j)
  end do
end do
do j=2,n-1
  do i=2,n-1
    c(i,j)=a(i,j-1)
  end do
end do
```

```
do j=2,n-1
  do i=2,n-1
    b(i,j)=a(i,j)
    c(i,j)=a(i,j-1)
  end do
end do
```

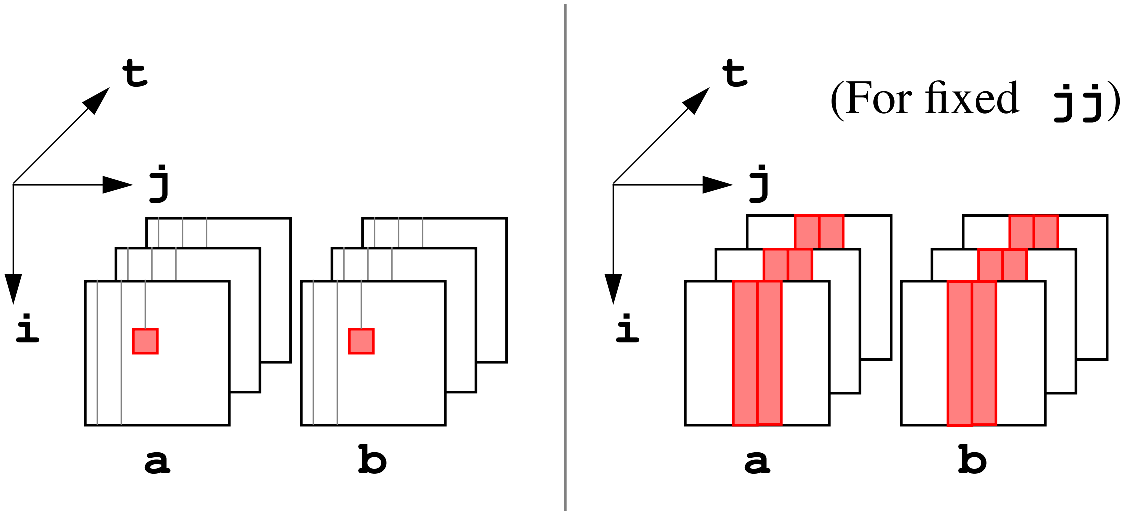


Benefits of Cache Partitioning (cont'd)

- *Tiling* to exploit reuse carried by outer loop:

```
do t=1,T
  do j=1,n
    do i=1,n
      ...=b(i,j)+a(i,j)
    end do
  end do
end do
```

```
do jj=1,n,s
  do t=1,T
    do j=jj,min(jj+s-1,n)
      do i=1,n
      ...=a(i,j)+b(i,j)
      end do
    end do
  end do
end do
```



Experimental Results

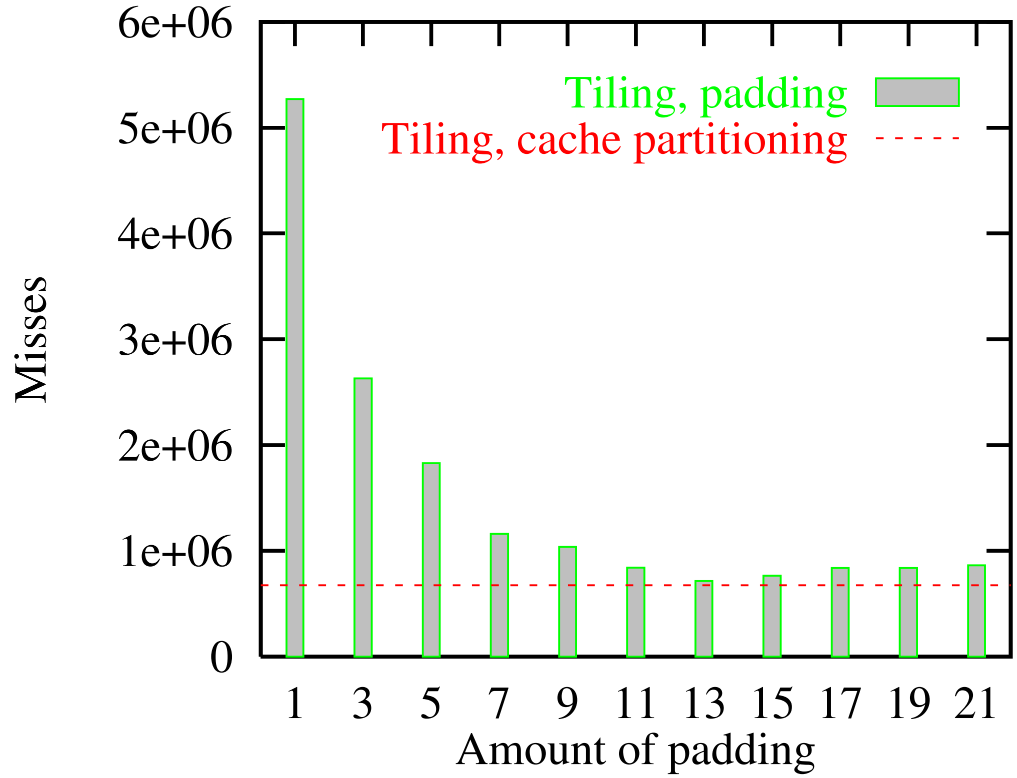
- Experiments on single processor of KSR multiprocessor:

Loop nest sequence	Loop nests	Arrays	Array size
Jacobi	2	2	500×500
`LL18`	3	9	256×256
`calc`	3	6	256×256

- Results to be reported:
  - cache partitioning vs. array padding
  - fusion with cache partitioning
  - tiling with cache partitioning

Experimental Results (cont'd)

- Cache partitioning vs. array padding for tiled version of **`LL18`**:



Experimental Results (cont'd)

- For *fusion*, measure cache misses and speedup:

Loop nest sequence	Predicted misses	Measured misses	Measured Speedup
Jacobi	62500	63243	1.34
`LL18`	49152	49677	1.26
`calc`	74305	77570	1.20

- For *tiling*, measure speedup, compare with upper bound:

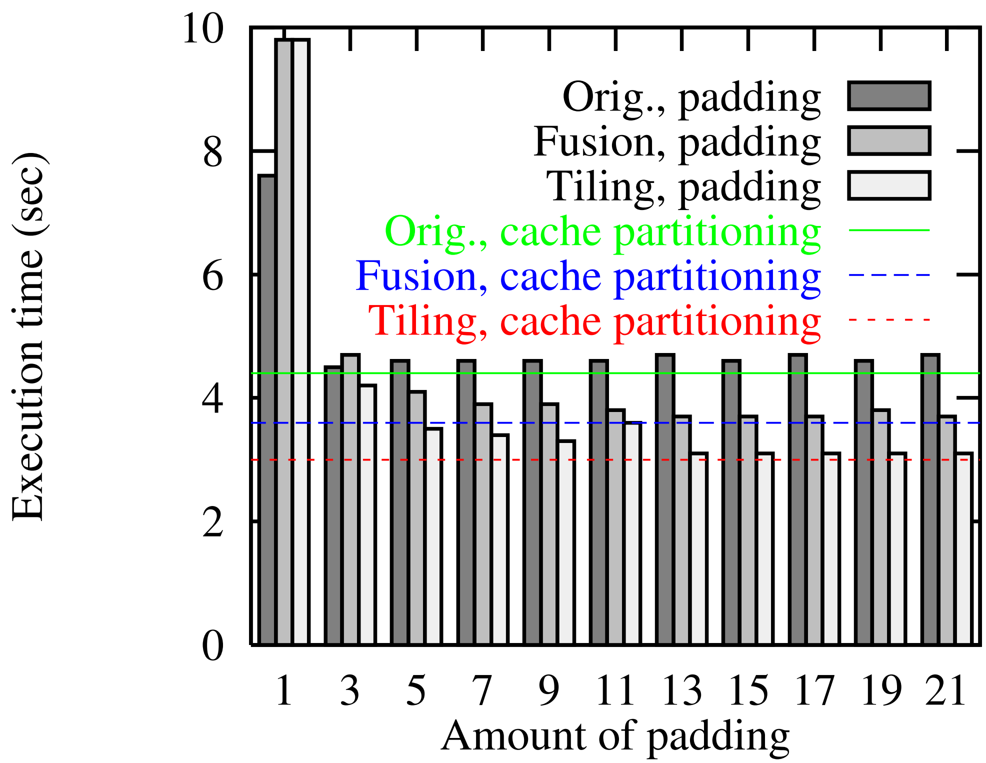
Loop nest sequence	Upper bound on speedup	Measured Speedup
Jacobi	2.51	2.27
`LL18`	1.77	1.55

  – upper bound determined by excluding memory time

  – measured speedup limited only by *loop skewing*

Experimental Results (cont'd)

- Experiments on one processor of SGI Challenge multiprocessor using **`LL18`**:



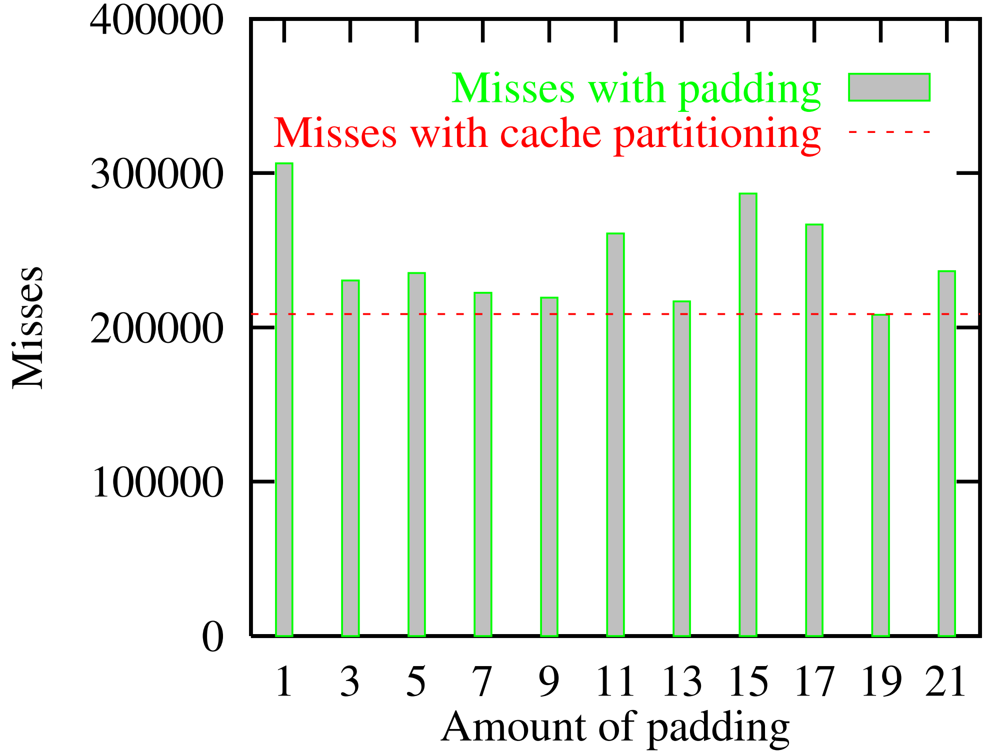
Experimental Results (cont'd)

- Parallel execution on KSR and Convex SPP-1000 multiprocessors

- Results to be reported:
  - cache partitioning vs. array padding
  - parallel execution of fused loops with cache partitioning
  - parallel execution of complete application with cache partitioning

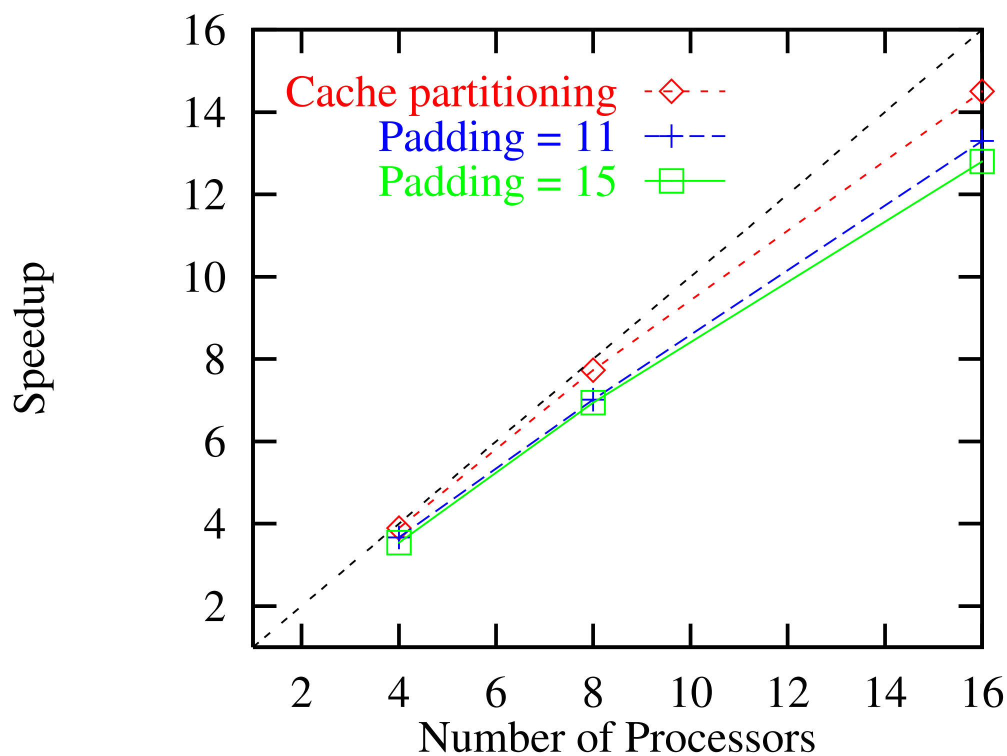
Experimental Results

- Cache partitioning vs. array padding, parallel execution of **`LL18`**:



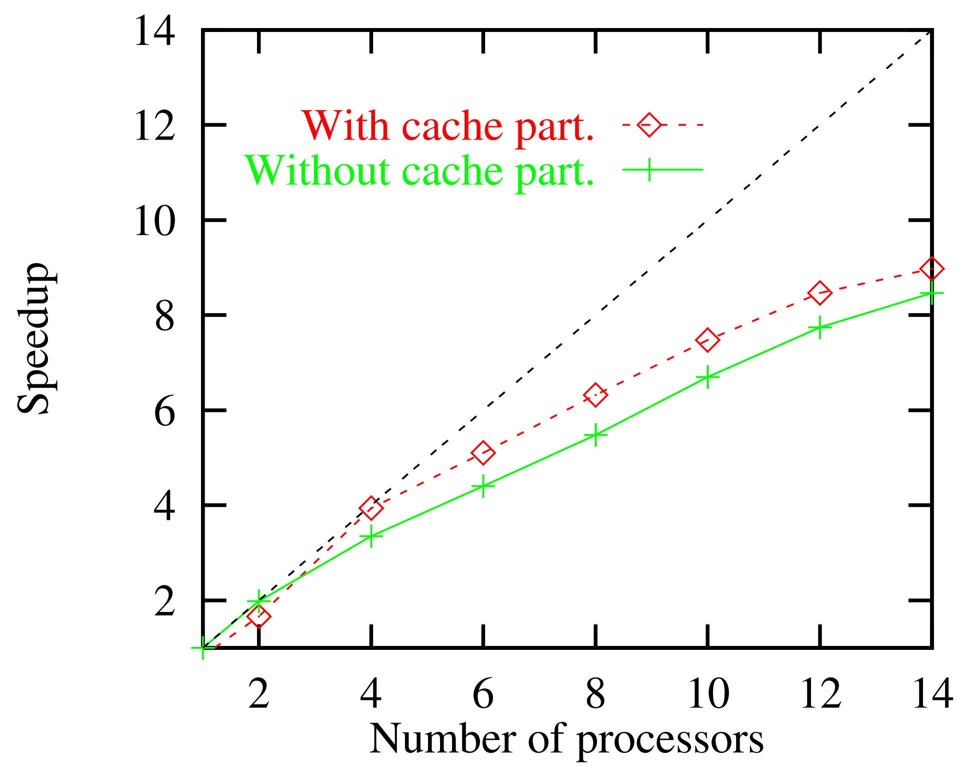
Experimental Results (cont'd)

- Parallel speedup for **`LL18`**:



Experimental Results (cont'd)

- SPEC92 benchmark **`swm256`** on Convex SPP-1000
- Performance improvement *due only to cache partitioning*:



Conclusions

- *Cache partitioning:* data layout to eliminate cache conflicts in parallel loops
- Avoids conflicts for data reuse carried by *outer* loops, not only inner loops
- Avoids trial-and-error approach of array padding
- Maximizes benefit of locality-enhancing loop transformations

Current Status and Future Work

- Implementation of cache partitioning in parallel compiler framework

- Future work:
  - extend to multiple levels of cache, set-associative caches
  - study interaction with data distribution for scalable multiprocessors