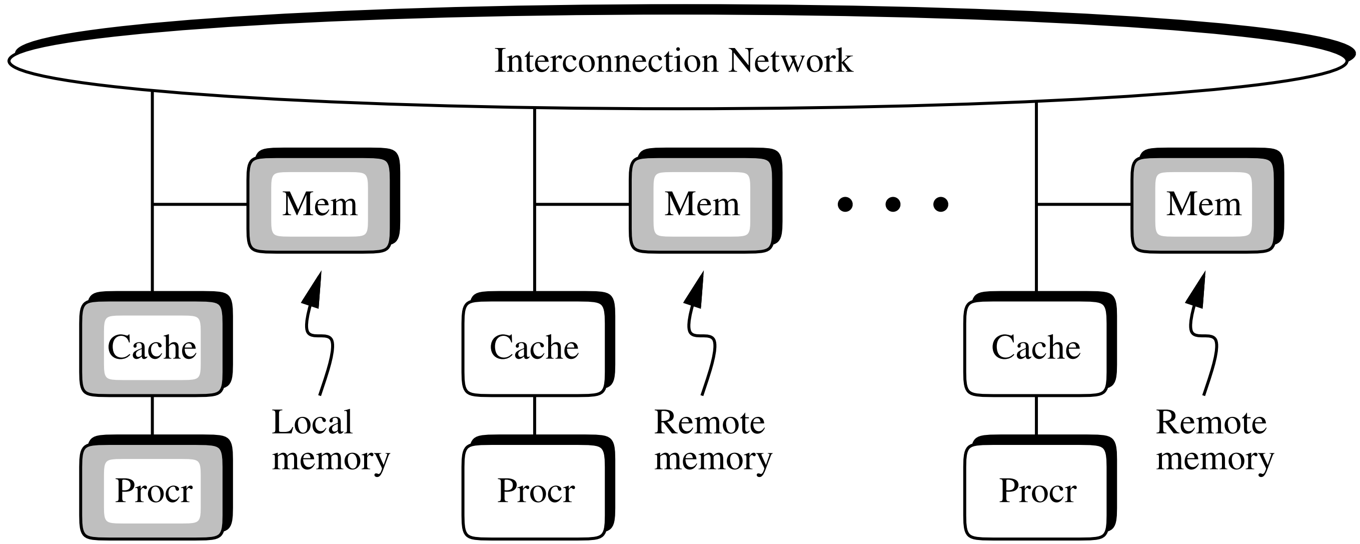
Fusion of Loops for Parallelism and Locality

Naraig Manjikian and Tarek S. Abdelrahman

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

15 August 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
```

Performance maximized by exploiting parallelism *and* enhancing locality

Locality Enhancement for Parallel Loops

- Locality enhancement for *individual* loops:
  - tiling/blocking for cache locality
  - unroll-and-jam for register locality

- Enhancing locality *between* parallel loops is still important

- *Our focus:* fusion to enhance locality *without* loss of parallelism
  - *shift-and-peel:* new technique to enable legal fusion/parallelization
  - *cache partitioning:* new technique to prevent cache conflicts
  - experimental results demonstrate utility/effectiveness

Outline

1. Loop fusion
2. Shift-and-peel
3. Cache partitioning
4. Experimental results
5. Conclusions

Problems with Loop Fusion

```
doall i = ...
 a[i] = ...
end do

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

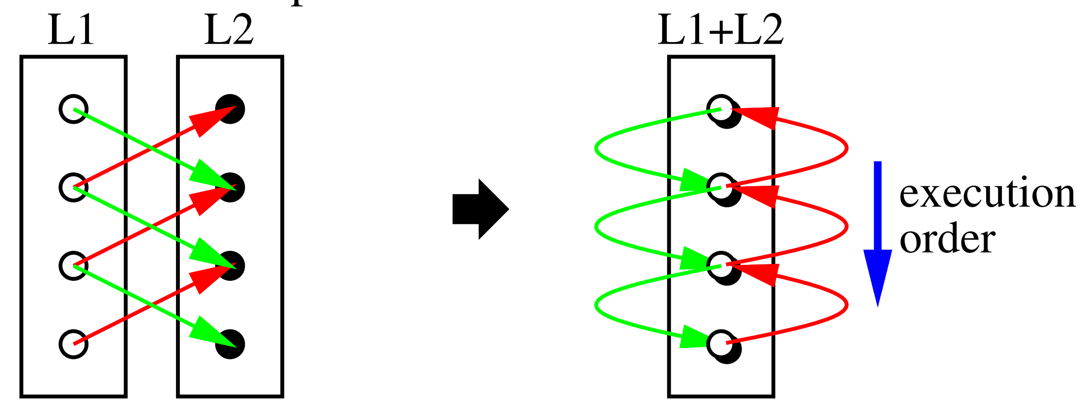
```
do i = ...

 a[i] = ...

 ... = a[i+1] + a[i-1]

end do
```

- “Fusion-preventing” dependences violated by iteration reordering
- Dependences which serialize loop execution after fusion

- Increase in cache conflicts after fusion:
  - more arrays, more data to be cached for reuse

Shifting

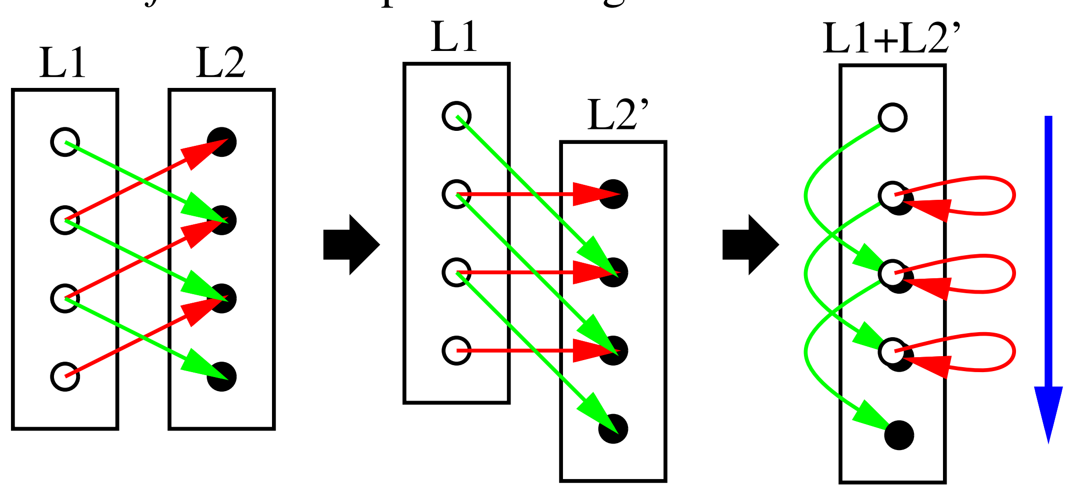
- Fusion may violate data dependences:



Dependences before fusion

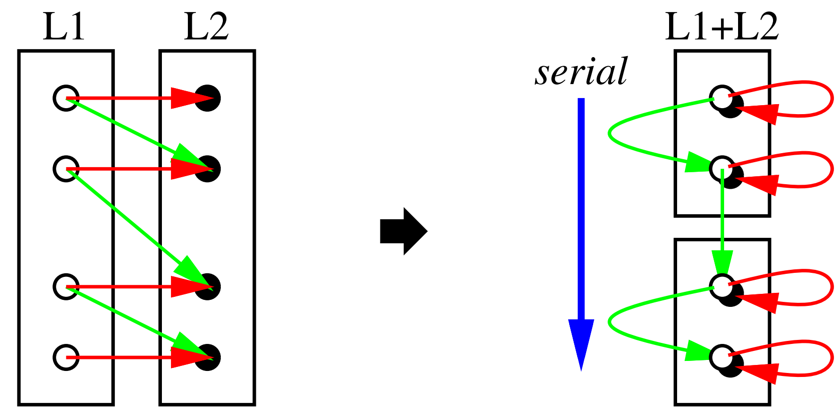
Illlegal fusion

- Our solution*: shift* iteration spaces for legal fusion:



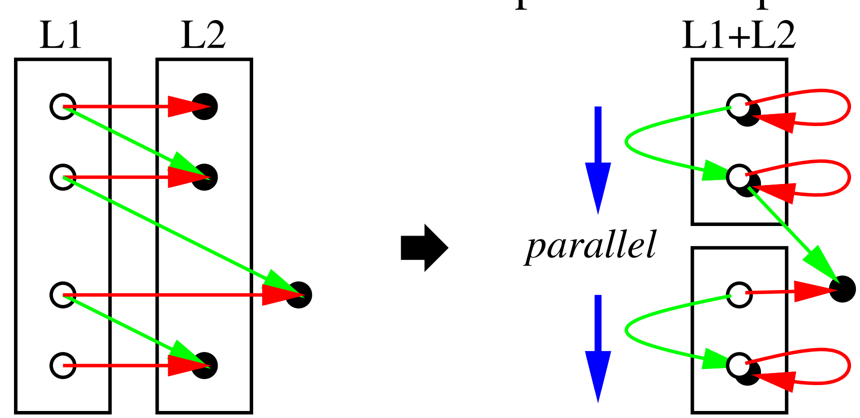
Peeling

- Dependences may serialize fused loop:



Fusion which results in a serial loop

- Our solution: *peel* sink iteration of cross-processor dependence:



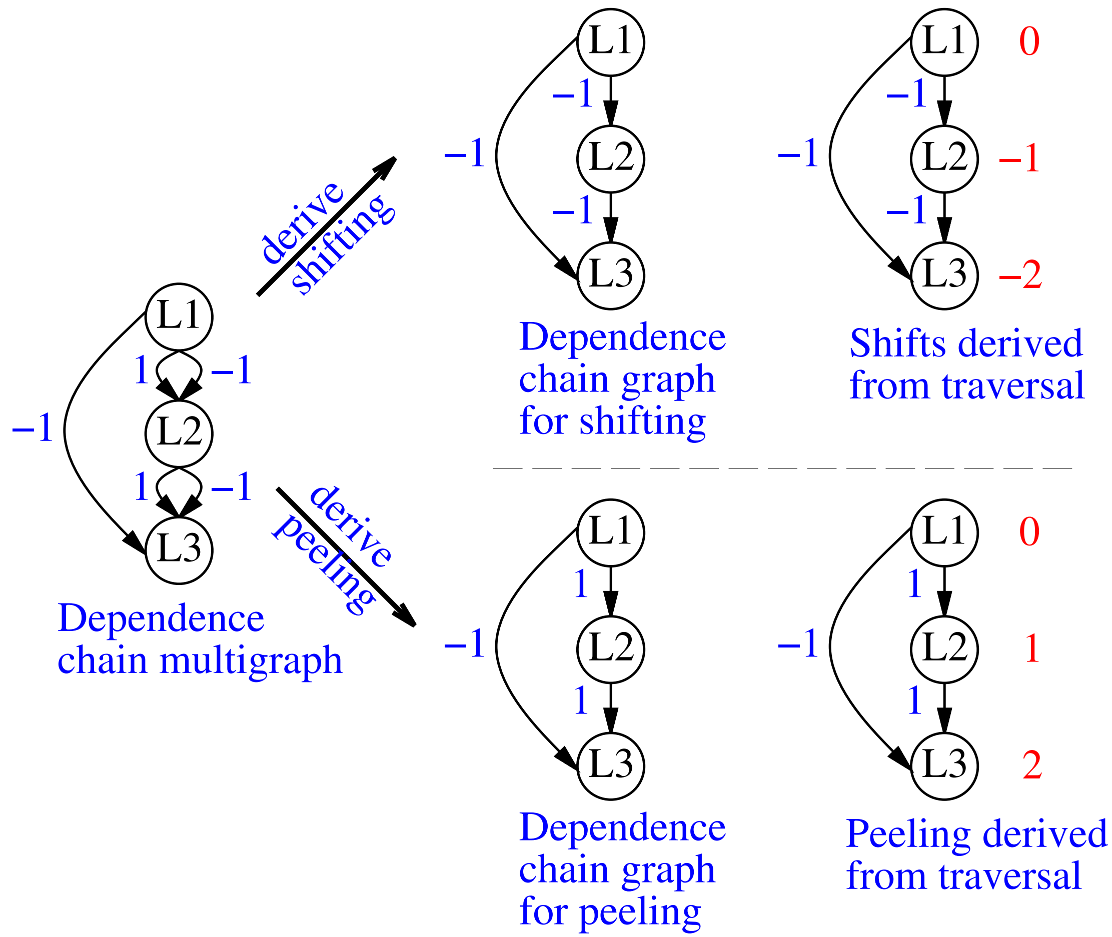
Peeling sink of cross–processor dependence
to permit parallel execution of fused loop

Derivation of Shift-and-Peel

- Use dependence distances between loop nests:

```
L1: do i=2,n−1
       a[i] = b[i]
    end do

L2: do i=2,n−1
       c[i] = a[i+1]+a[i−1]
    end do

L3: do i=2,n−1
        d[i] = c[i+1]+c[i−1]+a[i+1]
     end do
```

Loop nest sequence



Implementation of Shift-and-Peel

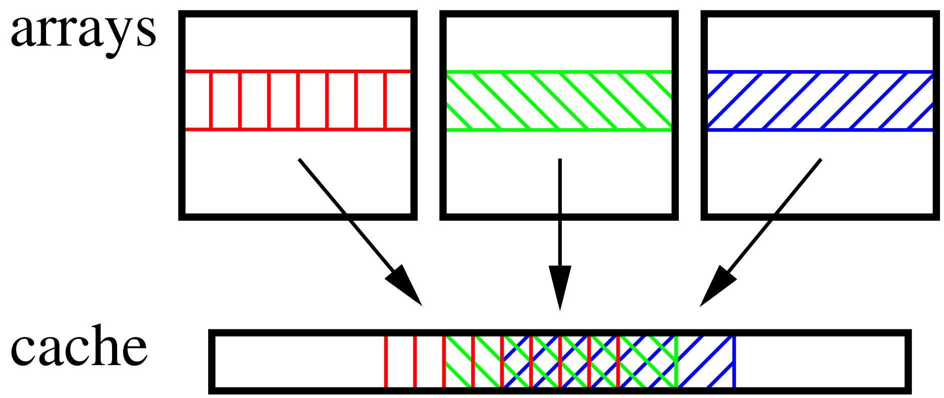
- Use *strip-mining* to implement fusion with shift-and-peel:

```
do ii=istart,iend,s
  do i=ii,min(ii+s-1,iend)
     a[i] = b[i]
  end do
  do i=max(ii-1,istart+1),min(ii+s-2,iend-1)
     c[i] = a[i+1]+a[i-1]
  end do
  do i=max(ii-2,istart+2),min(ii+s-3,iend-2)
     d[i] = c[i+1]+c[i-1]+a[i+1]
  end do
end do
```

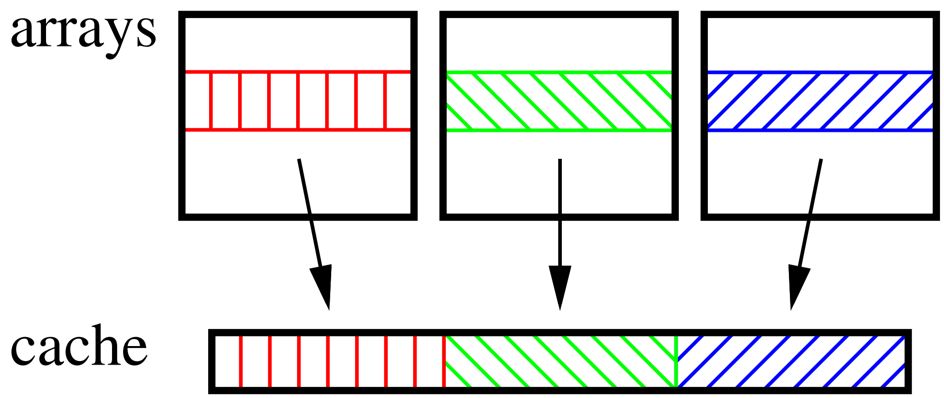
- Simpler than direct fusion, which requires guards, editing array expressions

Cache Partitioning to Eliminate Conflicts

- With fusion, more arrays referenced and more data must remain cached
- Cache conflicts occur when data mapping overlaps in cache:



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



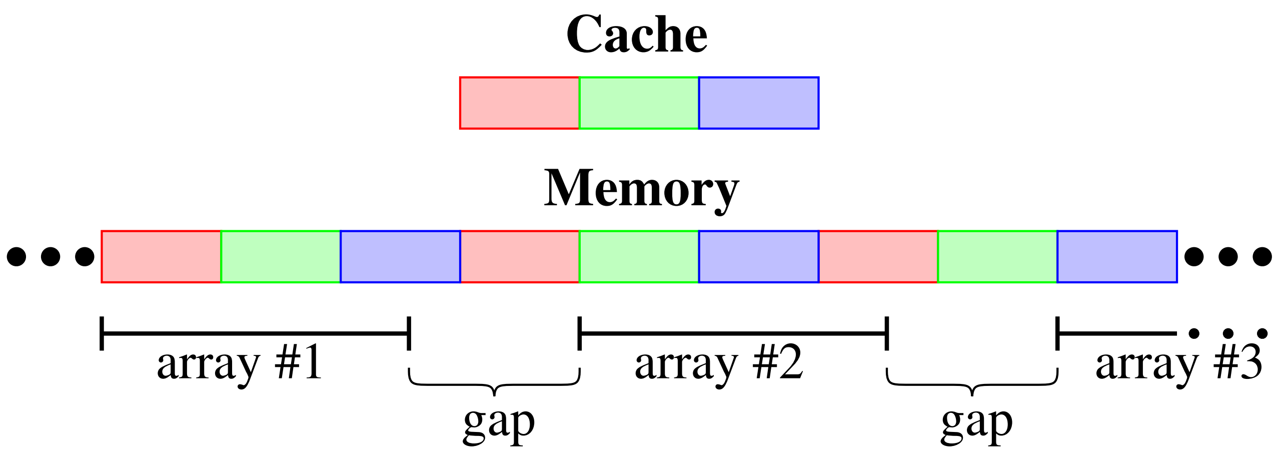
Cache Partitioning to Eliminate Conflicts (cont'd)

- Assumption: data accesses are compatible

Compatible data accesses

Incompatible data accesses

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



Experimental Results

- Experiments conducted on 56-processor KSR2, 16-processor Convex SPP
- Loop nest sequences:
  - Kernel 18 from Livermore Loops (3 loop nests, 9 arrays)
  - **`calc`** subroutine from **`qgbox`** (5 loop nests, 6 arrays)
  - **`filter`** subroutine from **`hydro2d`** (10 loop nests, 8 arrays)
- Results to be presented:
  - shift-and-peel required for fusion/parallelization
  - cache partitioning vs. array padding
  - performance of fusion with cache partitioning
  - shift-and-peel vs. replication [Cal87,AS93]

Experimental Results (cont'd)

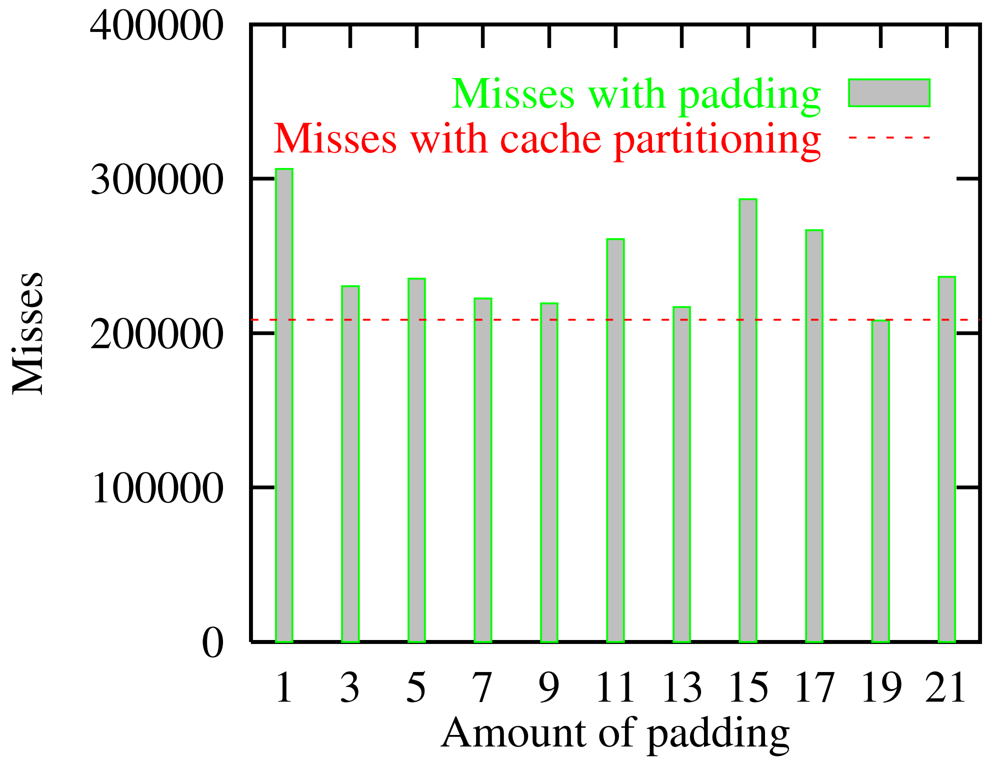
- Derivation of shift-and-peel for outermost loops:

Loop	LL18 shifts	LL18 peels	`calc` shifts	`calc` peels	`filter` shifts	`filter` peels
1	0	0	0	0	0	0
2	1	0	0	0	0	0
3	2	1	1	1	0	0
4			2	2	1	1
5			2	2	2	2
6					2	2
7					3	3
8					4	4
9					4	4
10					5	4

- Dependences "prevent" fusion/parallelization; *shift-and-peel required*

Experimental Results (cont'd)

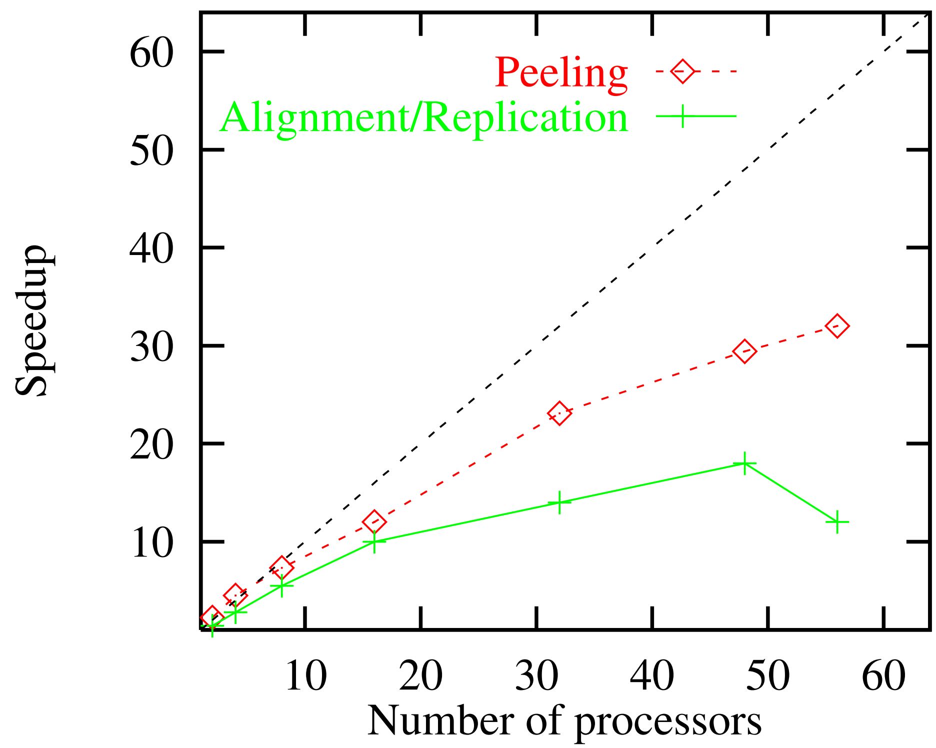
- Demonstrate importance of eliminating cache conflicts after fusion
- Cache partitioning compared with array padding for LL18:



- Original array sizes: 512×512

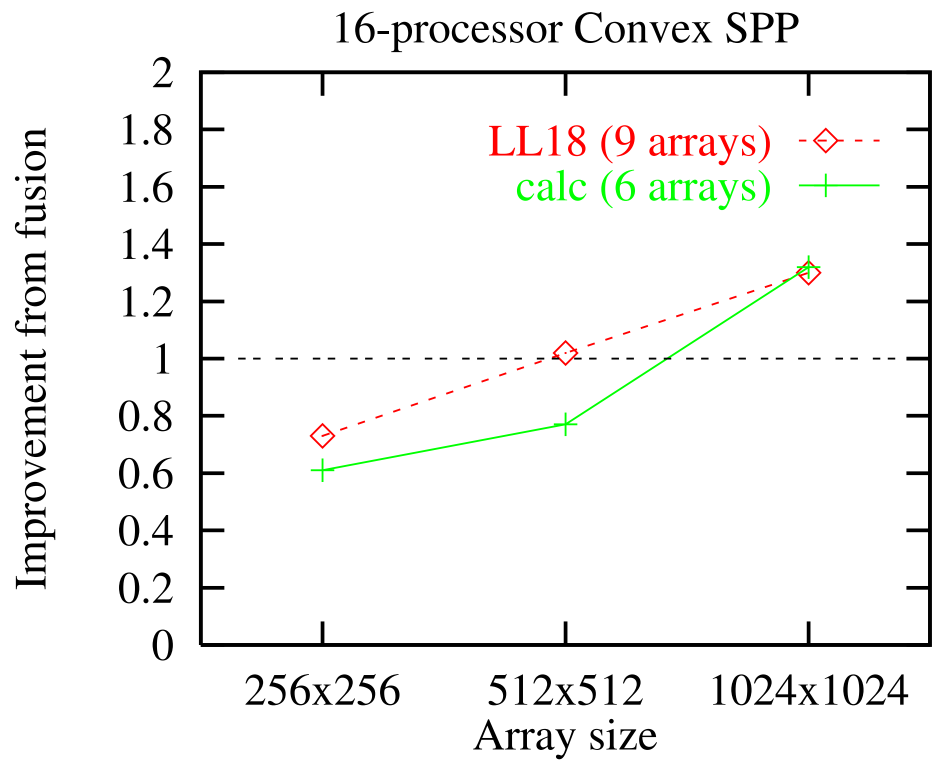
Experimental Results (cont'd)

- Comparison with alignment/replication for LL18 to demonstrate efficiency:



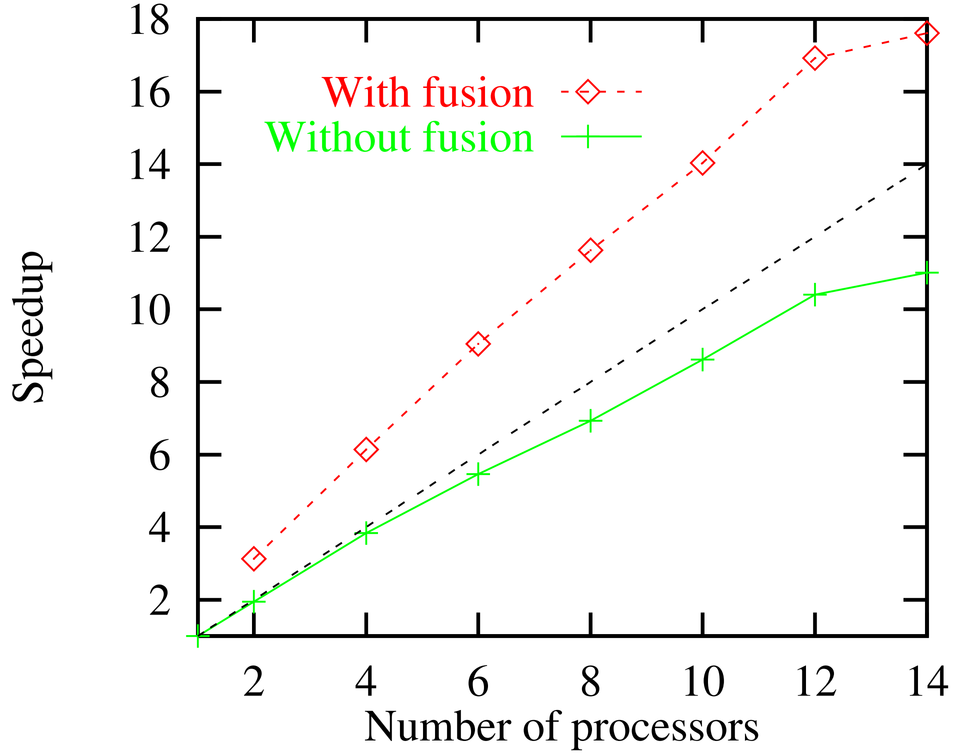
Experimental Results (cont'd)

- Improvement as problem size is varied for fixed number of processors:



Experimental Results (cont'd)

- Results for **`filter`** on Convex (array sizes are 1602×640):



Conclusions

- *Shift-and-peel* and *cache partitioning:* maximize locality/parallelism
- Shift-and-peel *required* for representative code; efficient technique
- Cache partitioning superior to array padding in avoiding cache conflicts
- Profitability of fusion: data size vs. cache size

Current Status and Future Work

- Compiler implementation of shift-and-peel and cache partitioning

- Further work:
    - more evidence of need for shift-and-peel to fuse/parallelize
    - evaluation of additional applications on more platforms