Tiling Optimizations for 3D Scientific Codes

PLDI'00?

Gabriel Rivera

Chau-Wen Tseng

Exploiting Reuse

- Memory hierarchy
- Compiler transformations

  Loop permutation:

```
do i = 1,N
  do j = 1,N
    = A(i,j)
```

```
do j = 1,N
  do i = 1,N
    = A(i,j)
```

  Padding:

```
real A(N), B(N)
do i=1,N
  S = S + A(i) + B(i)
```

```
real A(N), DUM(PAD), B(N)
do i=1,N
  S = S + A(i) + B(i)
```

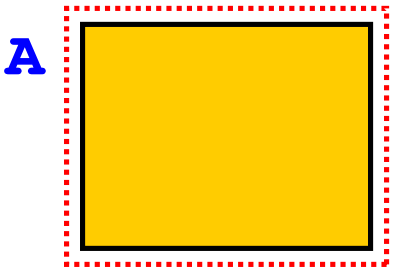
- Spatial reuse / temporal reuse

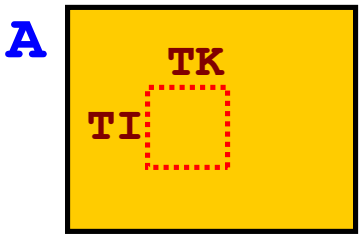
Tiling

- Move reuses closer in time

```
do J=1,N
 do K=1,N
  do I=1,N
   C(I,J) = C(I,J) +
    A(I,K)* B(K,J)
```

```
do KK=1,N,TK
 do II=1,N,TI
  do J=1,N
   do K=KK,min(KK+TK-1,N)
    do I=II,min(II+TI-1,N)
     C(I,J) = C(I,J) +
      A(I,K)* B(K,J)
```

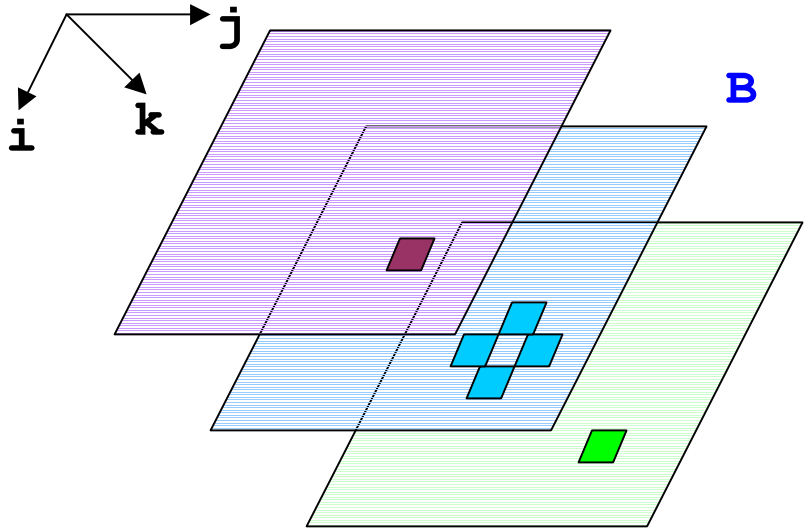




- Current methods
  - 2D, linear algebra codes
  - Copying avoids conflicts

3D Scientific Codes

- Iterative PDE solvers




```
real A(N,N,N), B(N,N,N)

do k=2,N-1
 do j=2,N-1
  do i=2,N-1
   A(i,j,k) =
     (B(i-1,j,k)+B(i+1,j,k)+
      B(i,j-1,k)+B(i,j+1,k)+
      B(i,j,k-1)+B(i,j,k+1))/6
```

- Average values in 3 planes
- Group-temporal reuse

  2 planes must fit in cache

Tiling 3D Scientific Codes

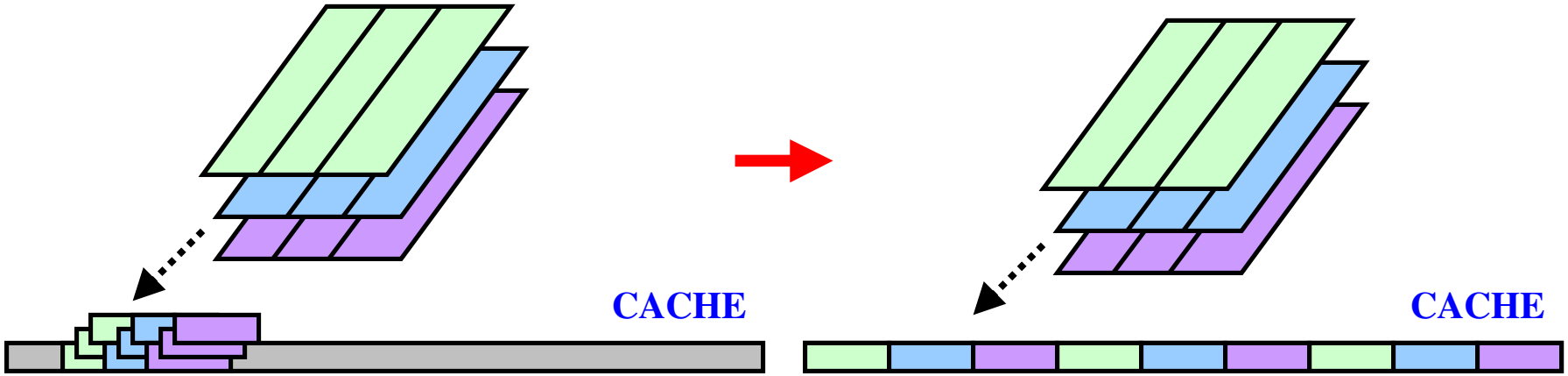
- **Reduce size of plane by tiling 2 loops**

```
do k=2,N-1
 do j=2,N-1
  do i=2,N-1
   A(i,j,k) = (B(i-1,j,k)+B(i+1,j,k)+B(i,j-1,k)+
               B(i,j+1,k)+B(i,j,k-1)+B(i,j,k+1))/6
```

```
do jj=2,N-1,TJ
 do ii=2,N-1,TI
  do k=2,N-1
   do j=jj,min(jj+TJ-1,N-1)
    do i=ii,min(ii+TI-1,N-1)
      A(i,j,k) = (B(i-1,j,k)+B(i+1,j,k)+B(i,j-1,k)+
                  B(i,j+1,k)+B(i,j,k-1)+B(i,j,k+1))/6
```

Avoiding Conflict Misses

- Want 3 planes to spread out on cache



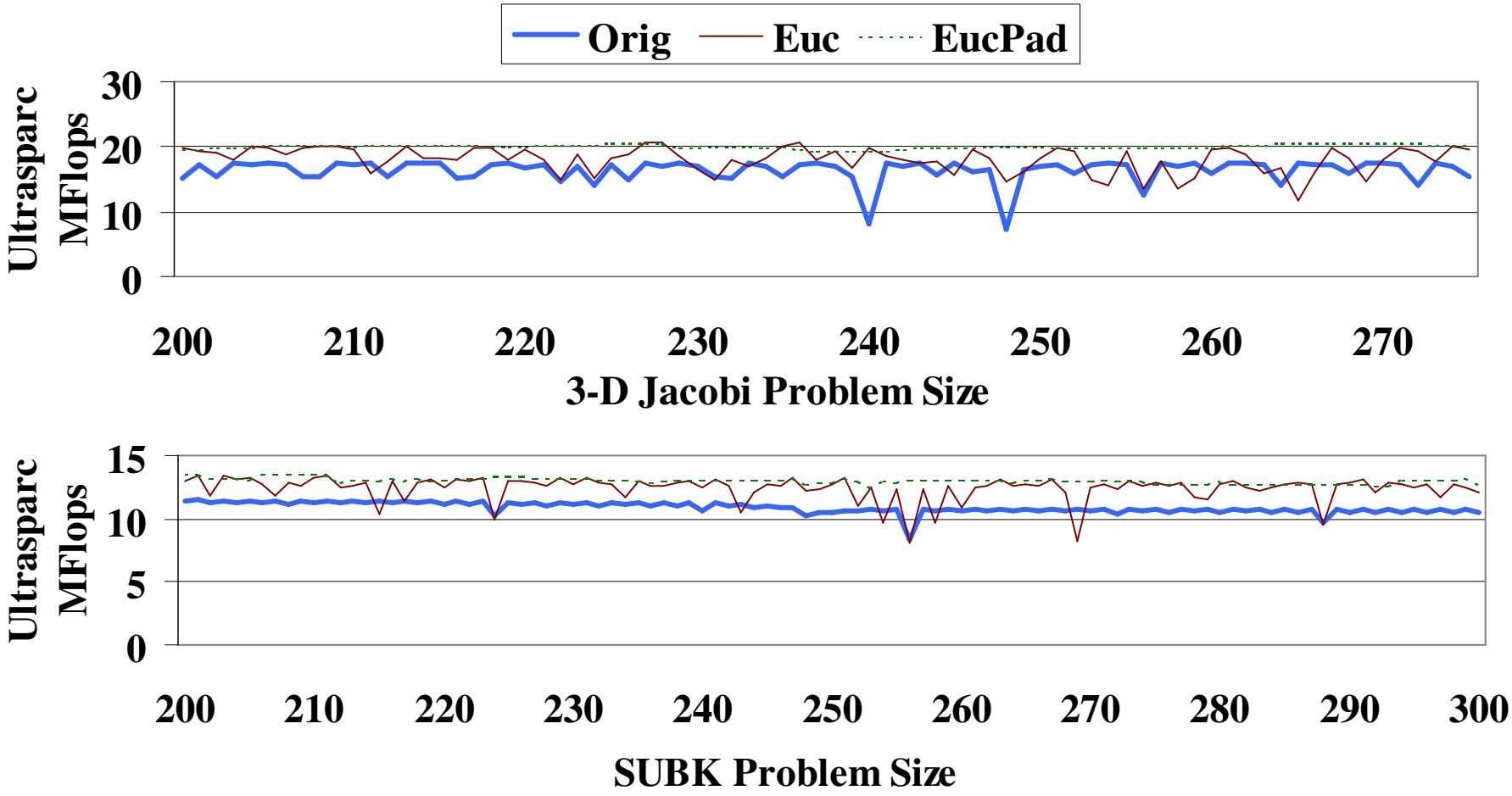
- Depends on array dimensions and cache parameters
- EUC Algorithm

  Compute non-self-interfering tiles
- EUCPAD Algorithm

  Consider nearby padded array sizes

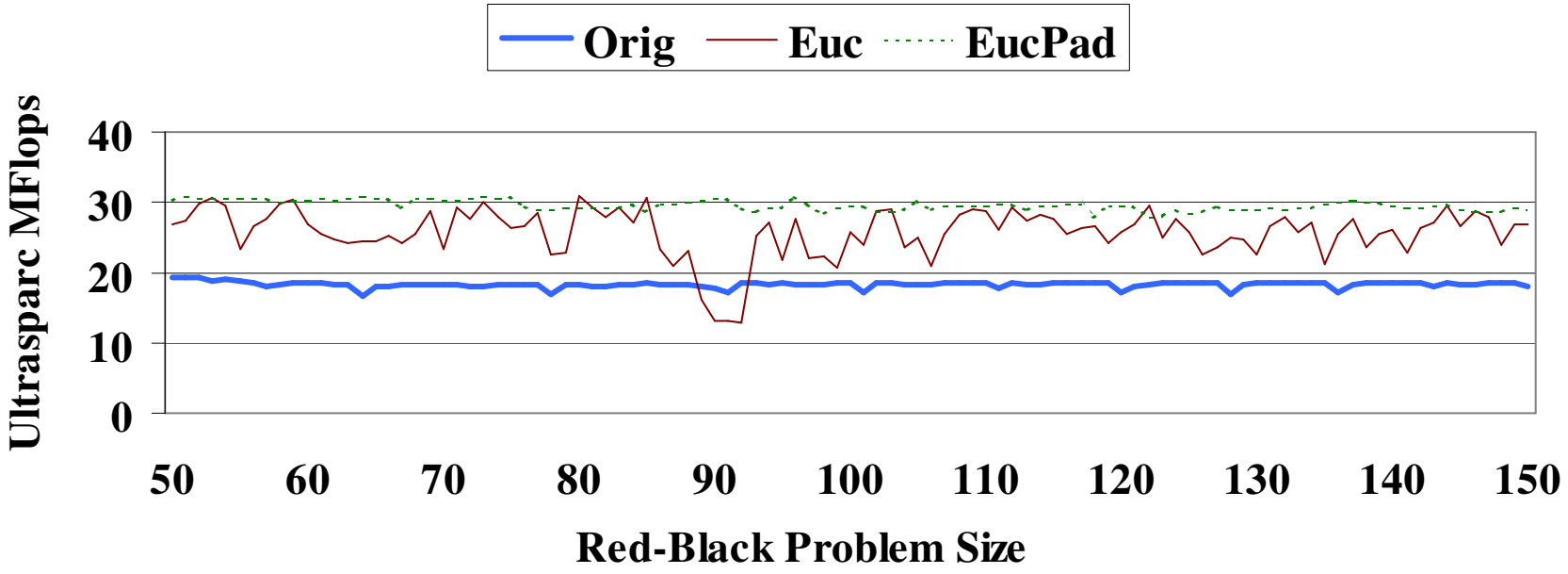
Experimental Results

- Padding stabilizes performance



Experimental Results

- Tiling also preserves spatial reuse
- Large, stable improvement by padding



Conclusions

- Large improvements possible
- Padding needed for stable performance
  - Difficult in larger applications
- Enable PDE solvers to better exploit memory hierarchy