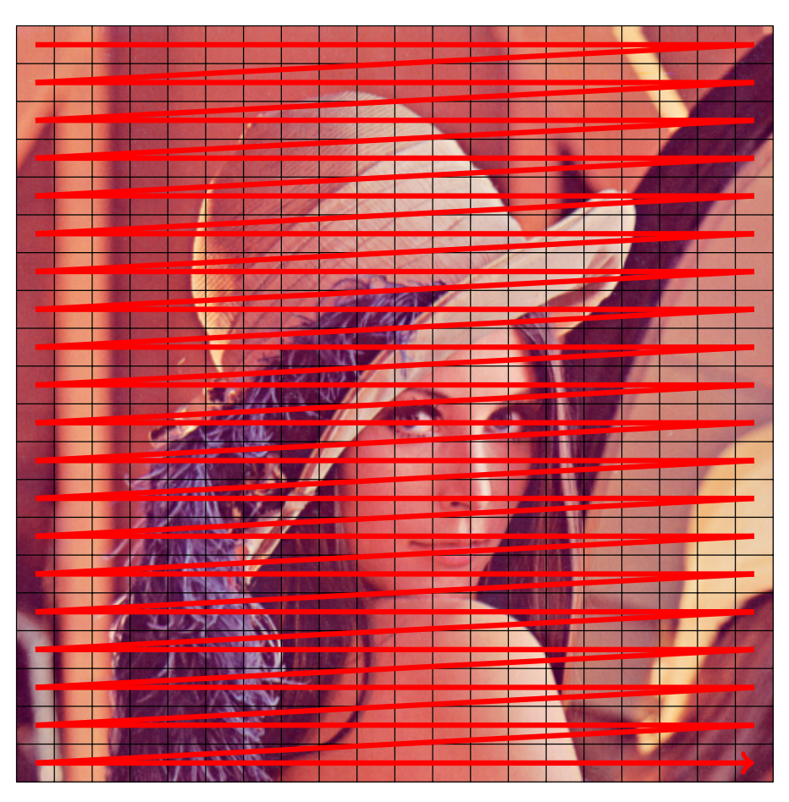
Automatic Source-to-Source Code Generation for Vector Hardware Accelerators

Serge Guelton, serge.guelton@telecom-bretagne.eu

LCPC'10 – Texas, USA – 7-9 October 2010

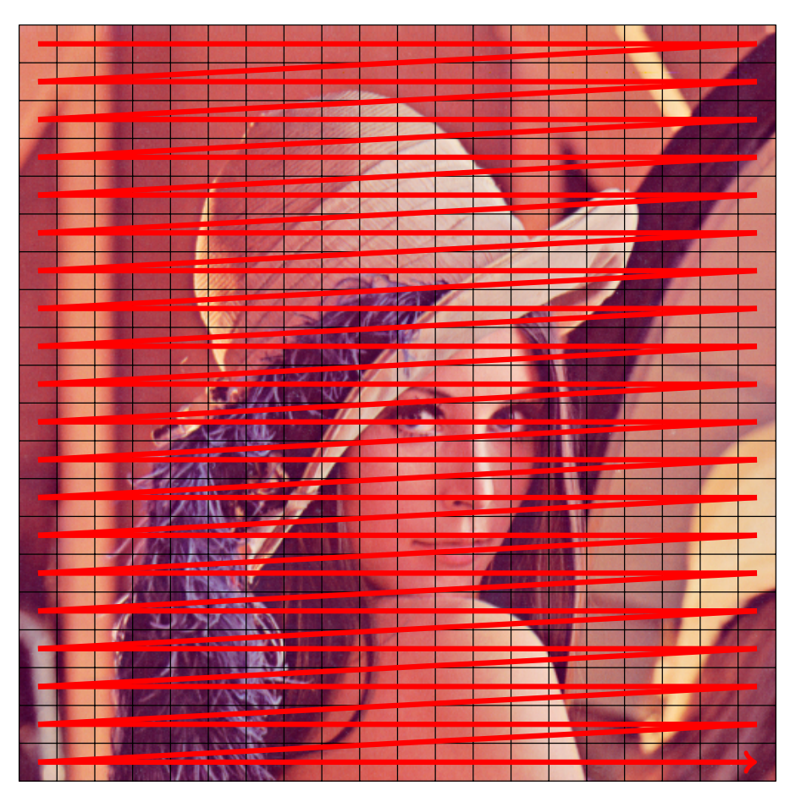
From high-level languages ...	to host and SIMD assembly code ...	using:
sequential	SIMD (fixed number of PE)	auto-paralleliz., sym. padding
unlimited memory	scratchpad memory	symbolic tiling
unique memory space	multiple memory spaces	statement isolation
homogeneous target	heterogeneous target	outlining
maintainable	unmaintainable	automatic transformations

Automatic Code Generation Scheme



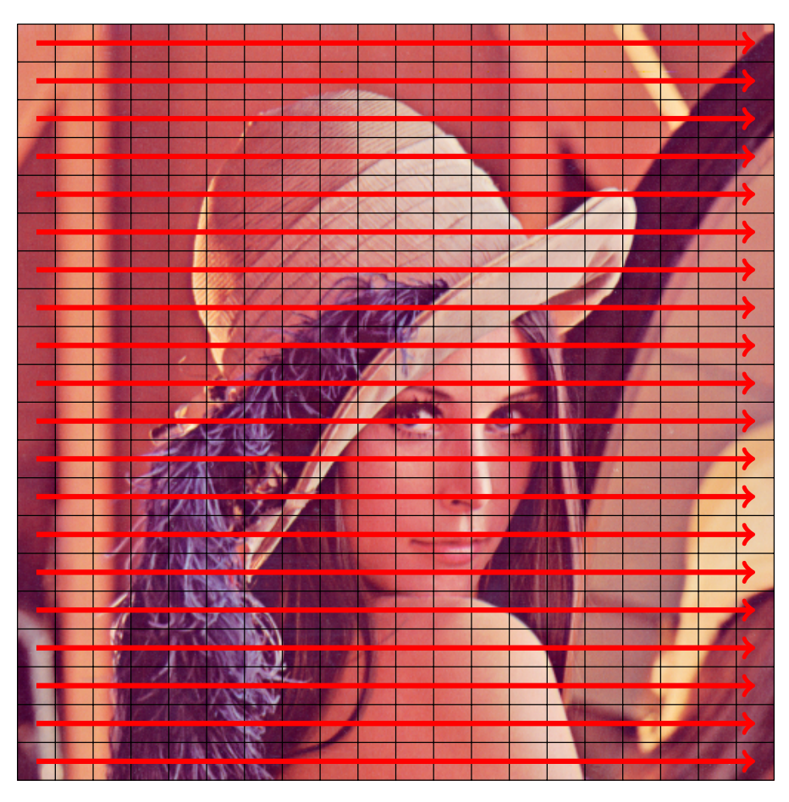
0 Sequential Code

```
for(i=0;i<20;i++)
 for(j=0;j<20;j++)
  lena[i][j]=
   (lena[i][j]+10)>255?255:(lena[i][j
    ]+10);
```



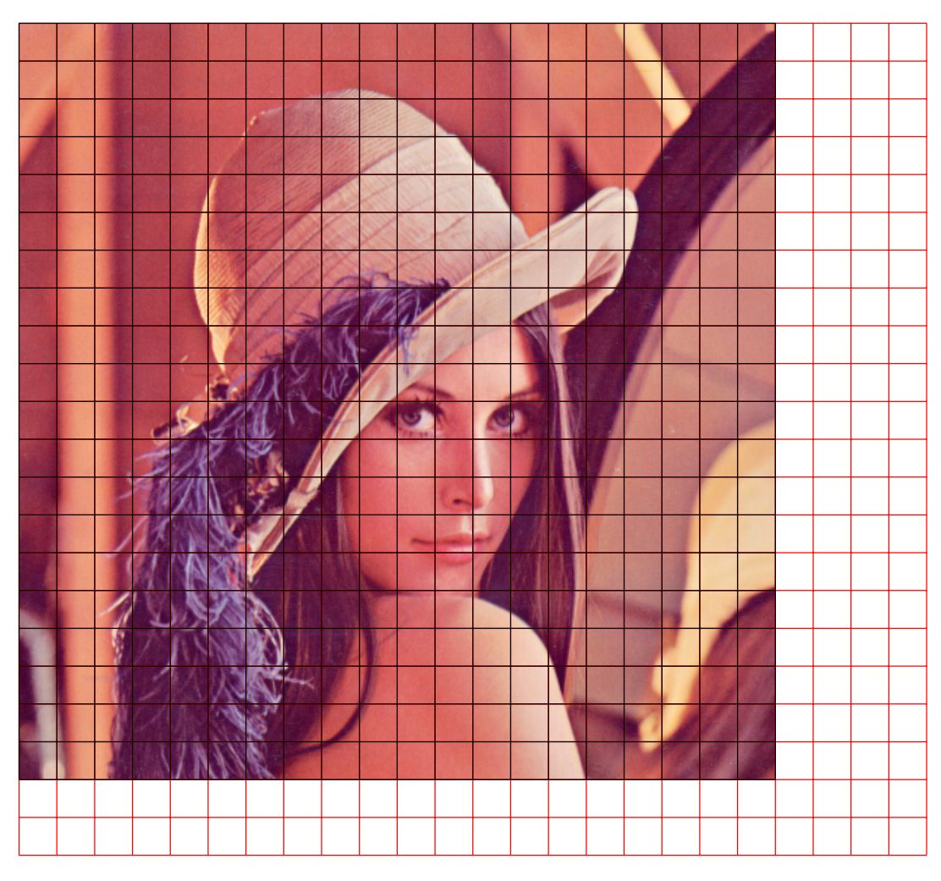
1 Pattern Recognition

```
for(i=0;i<20;i++)
 for(j=0;j<20;j++)
  lena[i][j]=adds(lena[i][j],10);
```



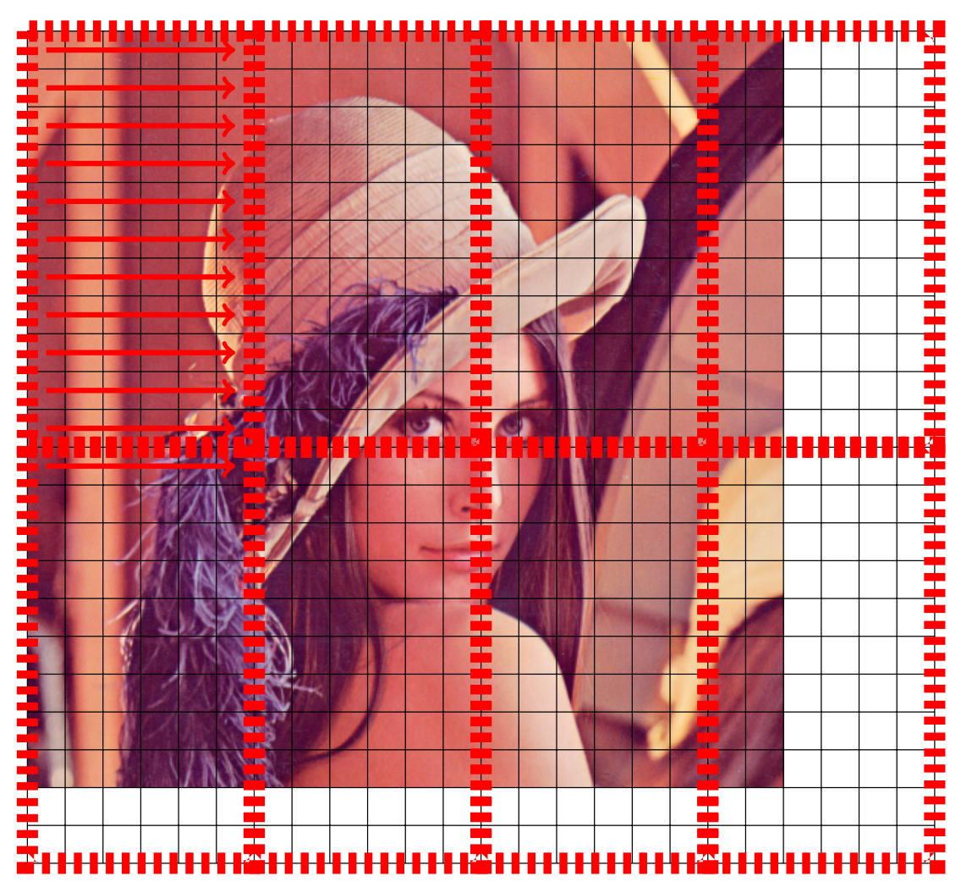
2 Detect Parallelism

```
#pragma omp parallel for
for(i=0;i<20;i++)
 for(j=0;j<20;j++)
  lena[i][j]=adds(lena[i][j],10);
```



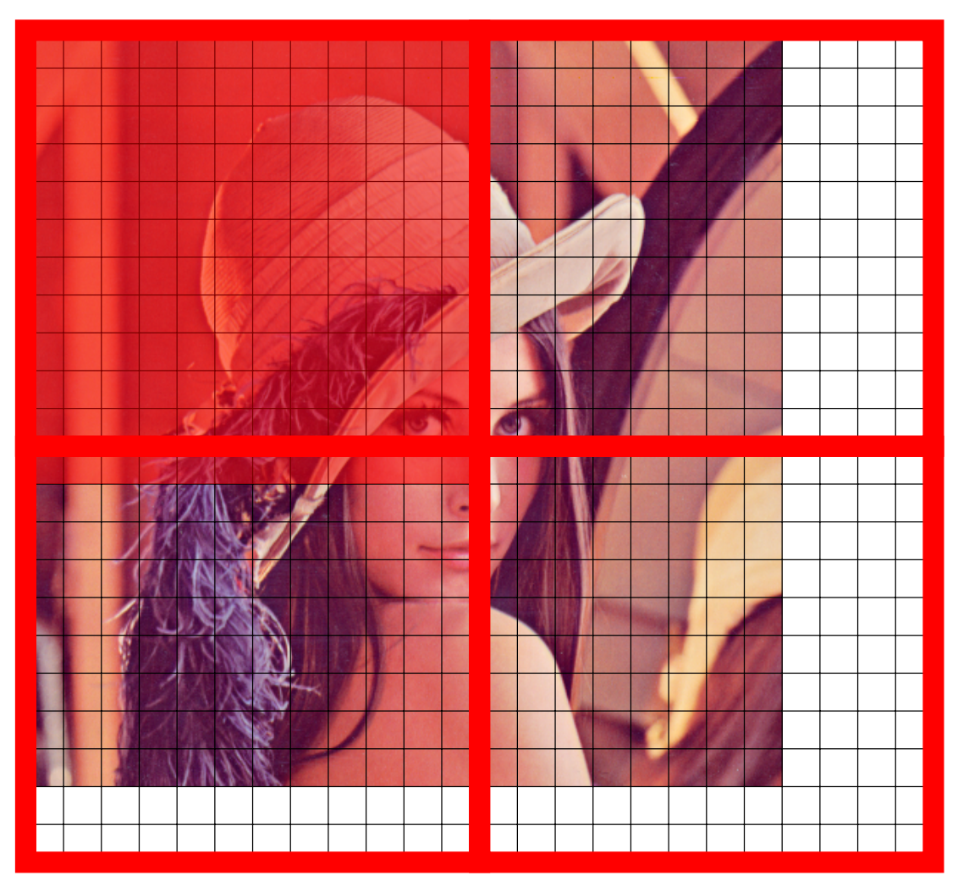
3 Symbolic Padding

```
int k, np=nb_proc;
expand(lena,k,np);
for(i=0;i<20/np*np;i++)
 for(j=0;j<20/k*k;j++)
  lena[i][j]=adds(lena[i][j],10);
```



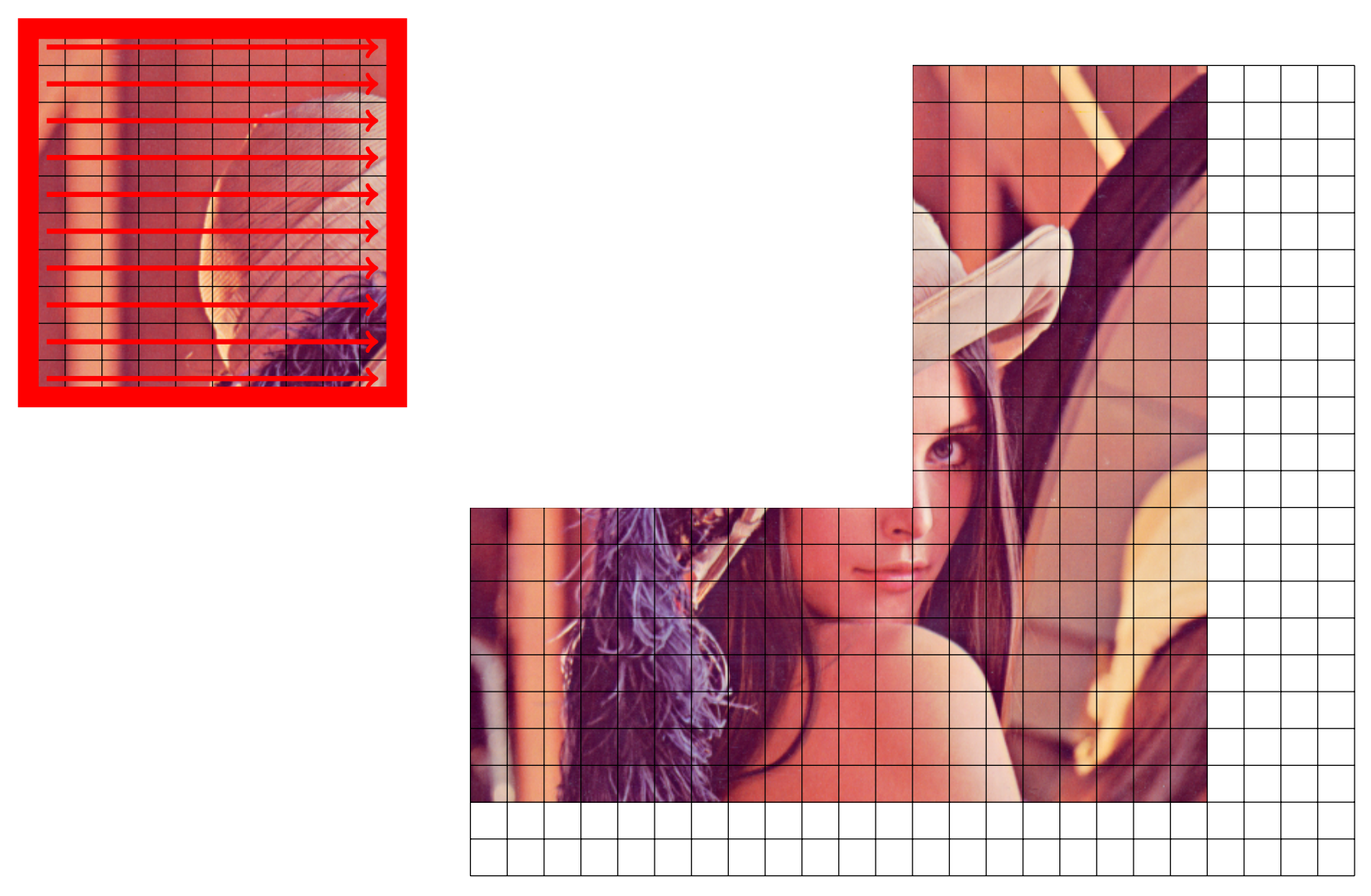
4 Symbolic Tiling

```
for(i=0;i<20/np*np;i+=np)
 for(j=0;j<20/k*k;j+=k)
  #pragma omp parallel for
  for(i_t=i;i_t<i+np;i_t++)
   for(j_t=j;j_t<j+k;j_t++)
    lena[i_t][j_t]=
     adds(lena[i_t][j_t],10);
```



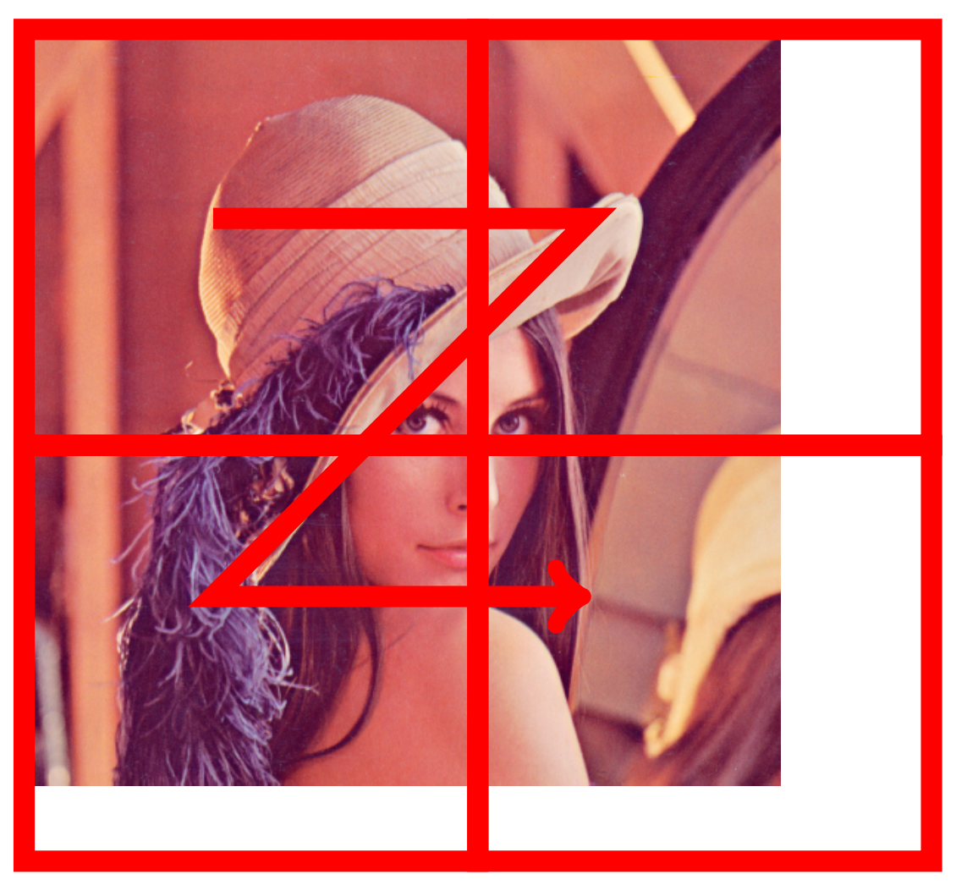
5 Effective Tiling

```
k=floor(proc_mem/np);
for(i=0;i<20/np*np;i+=np)
 for(j=0;j<20/k*k;j+=k)
  for(i_t=i;i_t<i+np;i_t++)
   for(j_t=j;j_t<j+k;j_t++)
    lena[i_t][j_t]=
     adds(lena[i_t][j_t],10);
```



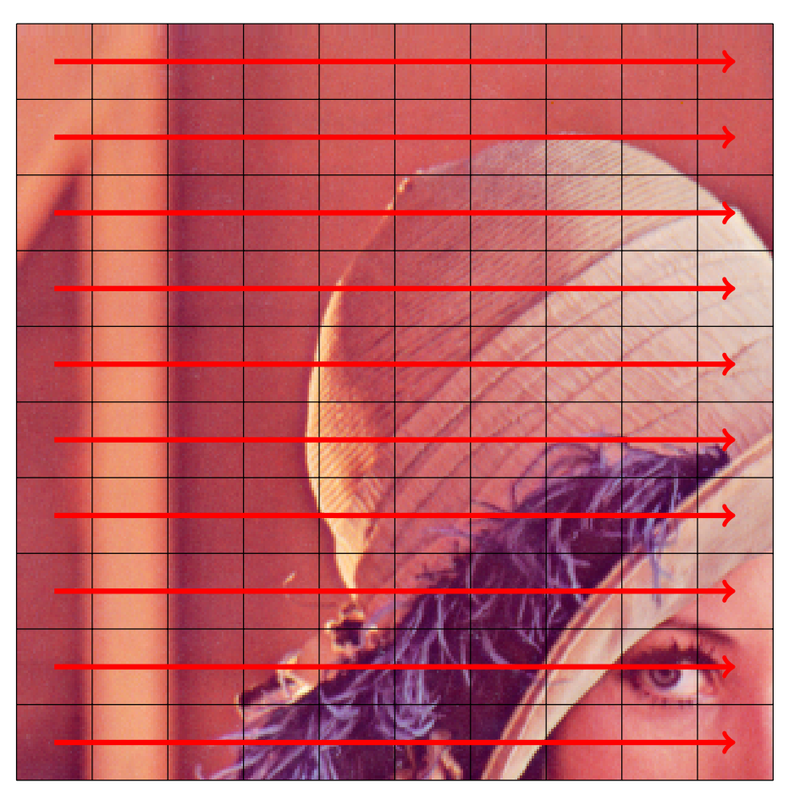
6 Statement Isolation

```
for(i=0;i<20/np*np;i+=np)
 for(j=0;j<20/k*k;j+=k) {
  char (*rl)[20/np*np][20/k*k];
  allocate(*rl,1,2D,np,k);
  copy(*rl,lena,2D,i,np,j,k);
  for(i_t=0;i_t<np;i_t++)
   for(j_t=0;j_t<k;j_t++)
    (*rl)[i][j]=
      adds((*rl)[i_t][j_t],10);
  copy(*rl,lena,2D,i,np,j,k);
  deallocate(rl);
 }
```



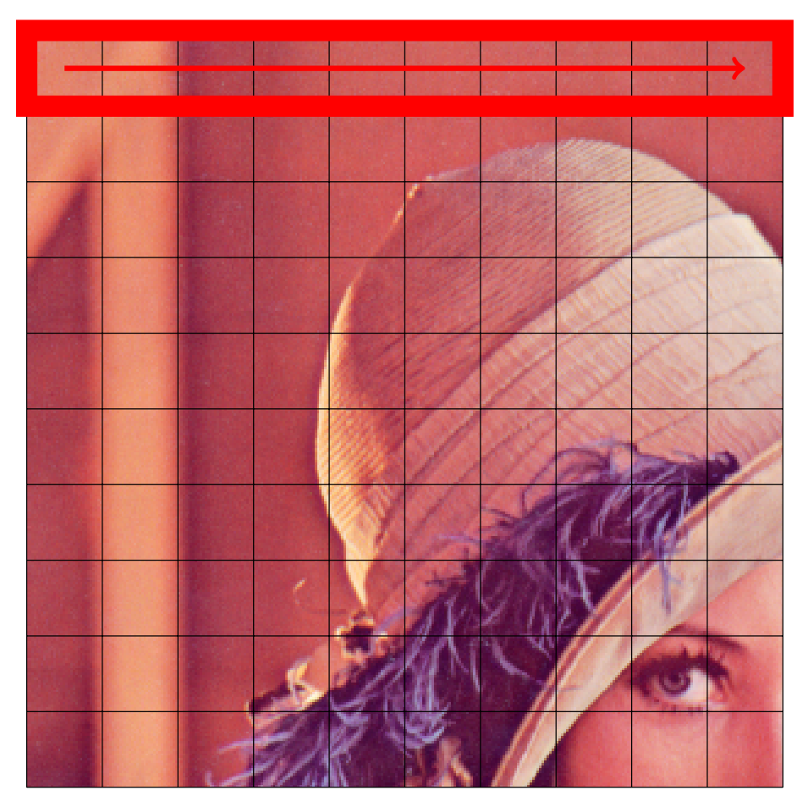
7 Outlining

```
for(i=0;i<20/np*np;i+=np)
 for(j=0;j<20/k*k;j+=k) {
  char (*rl)[20/np*np][20/k*k];
  allocate(*rl,sizeof(char),2D,np,k);
  copy(*rl,lena,2D,i,np,j,k);
  work(np,k,*rl);
  copy(*rl,lena,2D,i,np,j,k);
  deallocate(rl);
 }
```



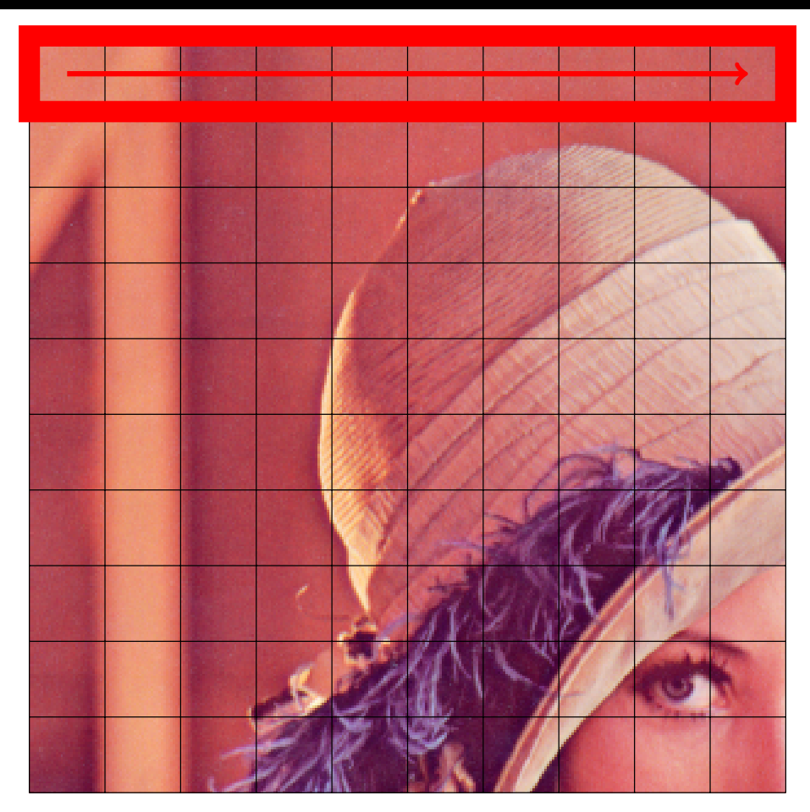
8 Outlined Code

```
void work(size_t np, size_t k,
    char rl[np][k]) {
size_t i,j;
#pragma parallel for
 for(i=0;i<np;i++)
  for(j=0;j<k;j++)
   rl[i][j]=
     adds(rl[i][j],10);
}
```



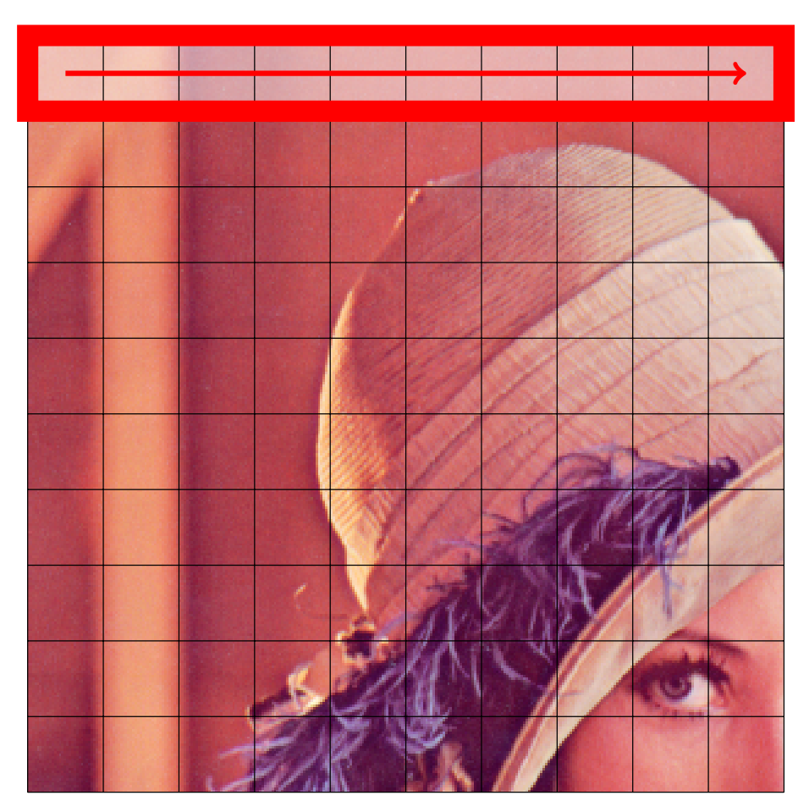
9 Kernel Code

```
void work(size_t np, size_t k,
    char rl[np][k]) {
#pragma parallel for
 for(i_t=0;i_t<np;i_t++)
   do_work(k,rl[i]);
}
void do_work(size_t k, char rl[k]) {
  size_t j;
  for(j=0;j<k;j++)
   rl[j]=adds(rl[j],10);
}
```



10 Simplified Kernel

```
void do_work(size_t k, char rl[k]) {
  size_t j=0;
  while(j<k) {
   *(rl+j)=
     adds((*(rl+j),10));
   j++;
  }
}
```



11 Kernel Asm

```
sub do_work
 im=FIFO1 || ||                 || ||
          || ||                 || || do_N1
          || || P,re(1) = adds(im1,10)
 im=im+P  || ||                 || ||
 im=im+E  || ||                 || ||
          || ||                 || || loop
endsub
```

Serge Guelton [Télécom Bretagne] – François Irigoin [Mines ParisTech] – Ronan Keryell [HPC Project]