Small Scale Parallel Programming Analysis of Matrix-Matrix Multiply

Salvatore Filippone

salvatore.filippone@cranfield.ac.uk

An Example: Matrix product

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad C_{ij} = C_{ij} + \sum_{1 \le k \le n} A_{ik} B_{kj}$$

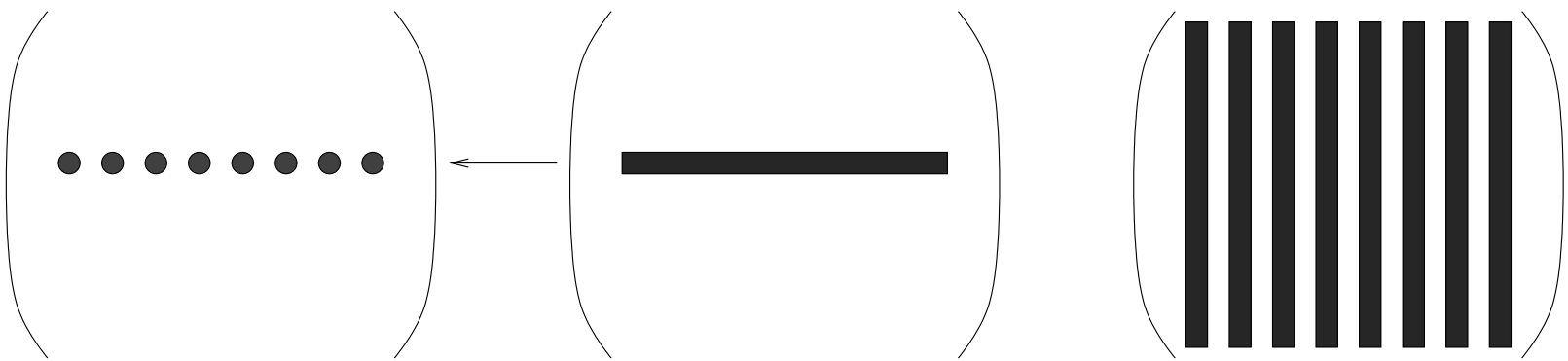
Gives rise to three nested loops

```
for i=1:n
  for j=1:n
    for k=1:n
       c(i,j)=a(i,k)*b(k,j)+c(i,j);
    end
   end
 end
```

Exercise: Compute the product $AB + C$ by employing all possible permutations of the loop nest, and compare the resulting speed; assume square matrices, and vary from $N = 1$ to $N = 1000$

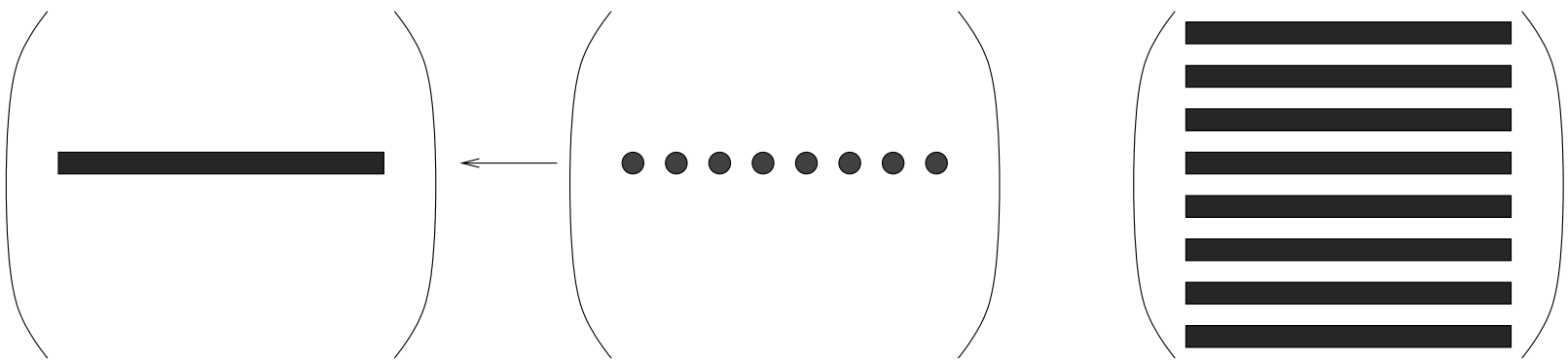
Matrix product: IJK

```
do i=1, n
 do j=1, n
   do k=1, n
    c(i,j)=a(i,k)*b(k,j)+c(i,j)
   end do
 end do
end do
```



Matrix product: IKJ

```
do i=1, n
 do k=1, n
   do j=1, n
    c(i,j)=a(i,k)*b(k,j)+c(i,j)
   end do
 end do
end do
```

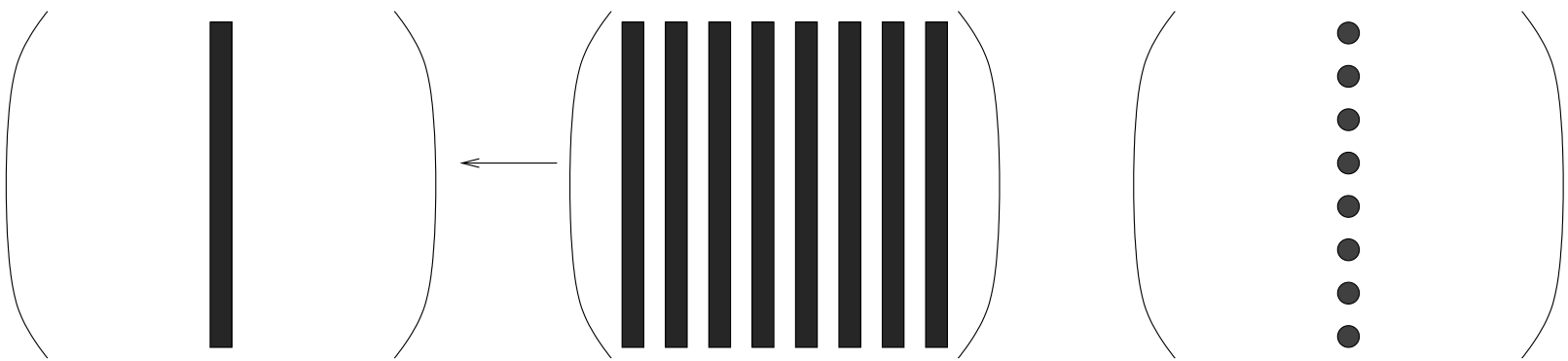


Matrix product: JIK

```
do j=1, n
 do i=1, n
   do k=1, n
    c(i,j)=a(i,k)*b(k,j)+c(i,j)
   end do
 end do
end do
```

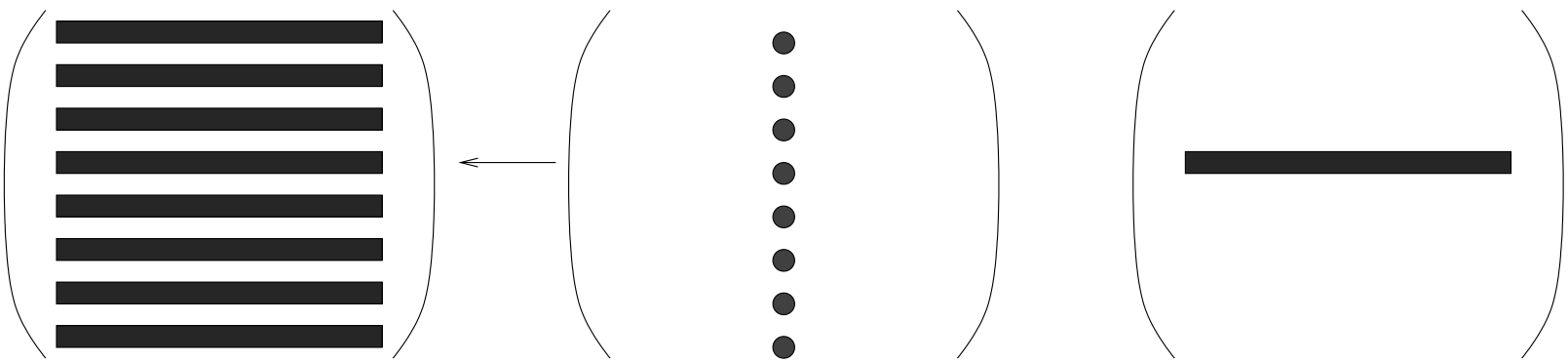
Matrix product: JKI

```
do j=1, n
 do k=1, n
   do i=1, n
    c(i,j)=a(i,k)*b(k,j)+c(i,j)
   end do
 end do
end do
```



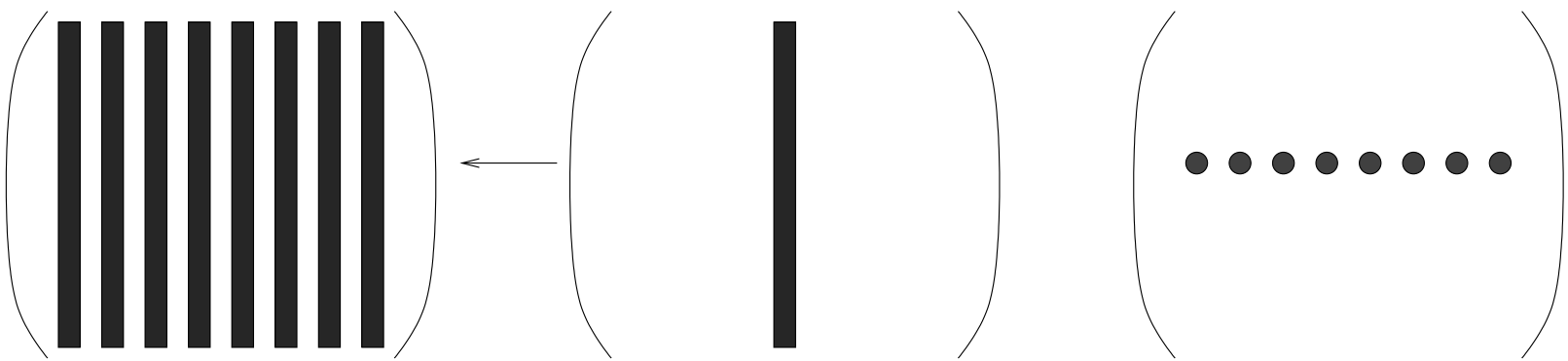
Matrix product: KIJ

```
do k=1, n
 do i=1, n
   do j=1, n
    c(i,j)=a(i,k)*b(k,j)+c(i,j)
   end do
 end do
end do
```

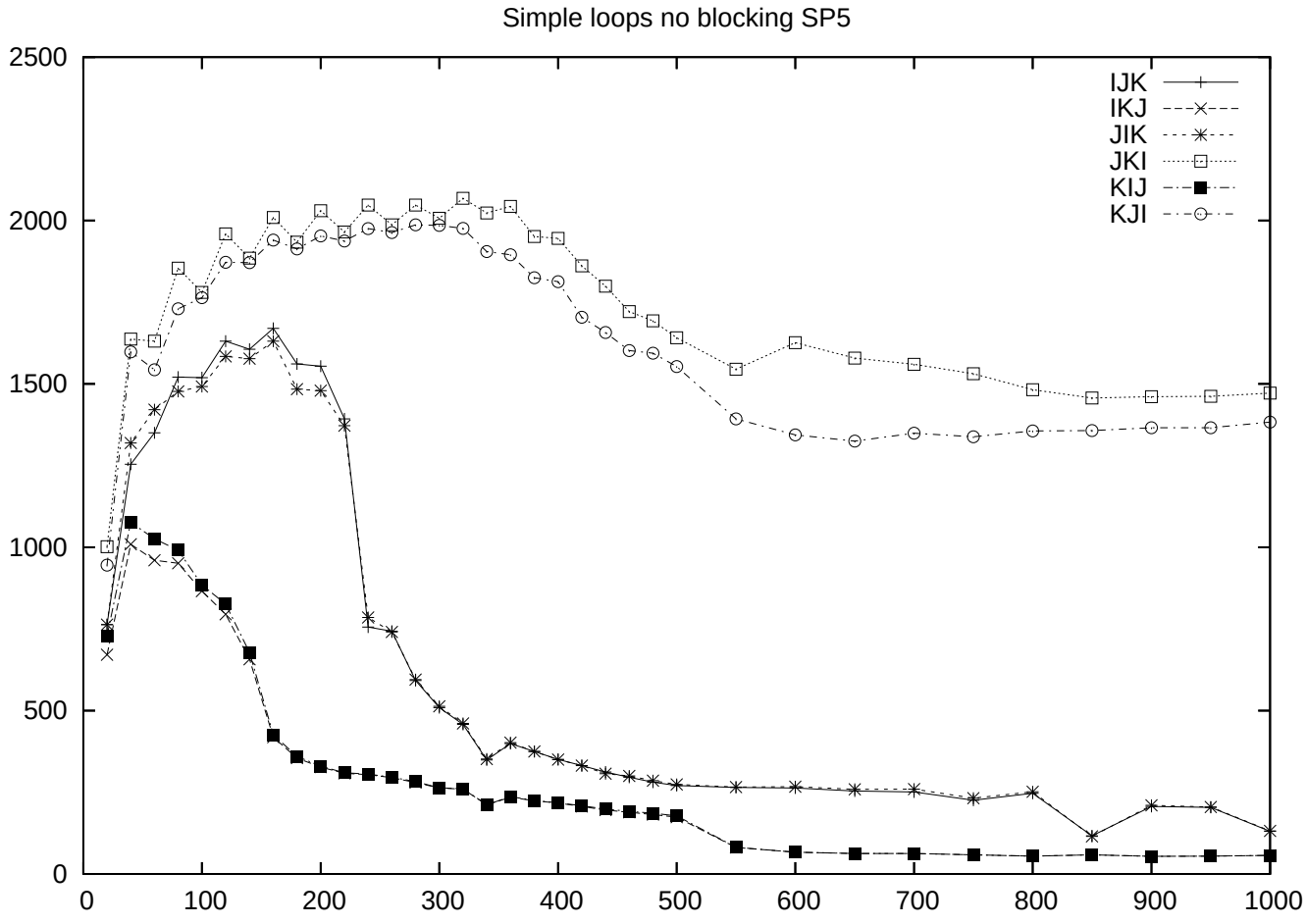


Matrix product: KJI

```
do k=1, n
 do j=1, n
   do i=1, n
    c(i,j)=a(i,k)*b(k,j)+c(i,j)
   end do
 end do
end do
```



Measurements



Matrix product 1: pipeline usage

```
do i=1, n
  do j=1, n
    do k=1, n
      c(i,j)=a(i,k)*b(k,j)&
      &       +c(i,j)
    end do
  end do
end do
```

```
20|                                   CL.118:
17| 000094 1     LFDU      fp0,gr11=a.rns9.(gr11,gr5,0)
17| 000098 1     LFDU      fp1,gr12=b.rns8.(gr12,8)
17| 00009C 1     FMA       fp7=fp7,fp0,fp1,fcr
 0| 0000A0 0     BCT       ctr=CL.118,taken=100%(100,0)
```

Execution time is dominated by LOAD instructions (2:1)

Matrix product 1: pipeline usage

```
do i=1, n
  do j=1, n
    do k=1, n
      c(i,j)=a(i,k)*b(k,j)&
      &       +c(i,j)
    end do
  end do
end do
```

```
17|                                  CL.120:
17| 000114 1     FMA       fp2=fp2,fp4,fp7,fcr
17| 000118 1     LFDU      fp12,gr11=a.rns9.(gr11,gr5,0)
17| 00011C 1     LFL       fp13=b.rns8.(gr12,8)
17| 000120 1     LFDU      fp4,gr11=a.rns9.(gr11,gr5,0)
17| 000124 1     LFL       fp7=b.rns8.(gr12,16)
17| 000128 1     LFDU      fp10,gr11=a.rns9.(gr11,gr5,0)
17| 00012C 1     LFL       fp11=b.rns8.(gr12,24)
17| 000130 1     FMA       fp5=fp5,fp12,fp13,fcr
17| 000134 1     LFDU      fp12,gr11=a.rns9.(gr11,gr5,0)
17| 000138 1     LFL       fp13=b.rns8.(gr12,32)
17| 00013C 1     FMA       fp0=fp0,fp4,fp7,fcr
17| 000140 1     LFDU      fp4,gr11=a.rns9.(gr11,gr5,0)
17| 000144 1     LFL       fp7=b.rns8.(gr12,40)
17| 000148 1     FMA       fp1=fp1,fp10,fp11,fcr
17| 00014C 1     LFDU      fp10,gr11=a.rns9.(gr11,gr5,0)
17| 000150 1     LFL       fp11=b.rns8.(gr12,48)
17| 000154 1     FMA       fp3=fp3,fp12,fp13,fcr
17| 000158 1     LFDU      fp12,gr11=a.rns9.(gr11,gr5,0)
17| 00015C 1     LFL       fp13=b.rns8.(gr12,56)
17| 000160 1     FMA       fp6=fp6,fp4,fp7,fcr
17| 000164 1     LFDU      fp4,gr11=a.rns9.(gr11,gr5,0)
17| 000168 1     LFDU      fp7,gr12=b.rns8.(gr12,64)
17| 00016C 1     FMA       fp8=fp8,fp10,fp11,fcr
17| 000170 1     FMA       fp9=fp9,fp12,fp13,fcr
 0| 000174 0     BCT       ctr=CL.120,taken=100%(100,0)
```

Execution time is STILL dominated by LOAD instructions (2:1) but the pipeline is better exploited. What did the compiler do????

Matrix product 2: Loop unrolling

Let us reorder the loops:

```
do j=1, n
  do i=1, n
    do k=1, n
      c(i,j)=a(i,k)*b(k,j) + c(i,j)
    end do
  end do
end do
```

In the innermost loop, variables I and J are fixed; Different iterations of the I loop use the same K and J values; can they proceed in parallel?

Matrix product 2: Loop unrolling

```
do j=1, n
  do i=1, n, 2
    do k=1, n
      c(i+0,j)=a(i+0,k)*b(k,j) + c(i+0,j)
      c(i+1,j)=a(i+1,k)*b(k,j) + c(i+1,j)
    end do
  end do
end do
```

Matrix product 2: Loop unrolling

```
do j=1, n
  do i=1, n, 2
    t0 = c(i+0,j)
    t1 = c(i+1,j)
    do k=1, n
      b0 = b(k,j)
      t0 = a(i+0,k)*b0 + t0
      t1 = a(i+1,k)*b0 + t1
    end do
    c(i+0,j) =  t0
    c(i+1,j) =  t1
  end do
end do
```

Ok, why stop at $\times 2$?

Matrix product 2: Loop unrolling

```
do i=i4+1,ib,4
   t0 = c(i+0,j)
   t1 = c(i+1,j)
   t2 = c(i+2,j)
   t3 = c(i+3,j)
   do k=1,kb
      b0 = b(k,j)
      t0 = t0 + a(i+0,k)*b0
      t1 = t1 + a(i+1,k)*b0
      t2 = t2 + a(i+2,k)*b0
      t3 = t3 + a(i+3,k)*b0
   end do
   c(i+0,j) =  t0
   c(i+1,j) =  t1
   c(i+2,j) =  t2
   c(i+3,j) =  t3
end do
```

```
55|                                  CL.170:
52| 1     FMA       fp9=fp5,fp0,fp7,fcr
52| 1     LFL       fp4=b.rns8.(gr27,16)
53| 1     FMA       fp5=fp11,fp7,fp1,fcr
54| 1     FMA       fp6=fp6,fp7,fp2,fcr
52| 1     LFDU      fp2,gr26=a.rns9.(gr26,gr30,0)
52| 1     LFL       fp0=b.rns8.(gr27,8)
55| 1     FMA       fp7=fp8,fp7,fp3,fcr
53| 1     LFL       fp10=a.rns9.(gr26,8)
54| 1     LFL       fp11=a.rns9.(gr26,16)
55| 1     LFL       fp8=a.rns9.(gr26,24)
52| 1     LFDU      fp1,gr26=a.rns9.(gr26,gr30,0)
52| 1     FMA       fp2=fp9,fp2,fp0,fcr
53| 1     LFL       fp13=a.rns9.(gr26,8)
54| 1     LFL       fp3=a.rns9.(gr26,16)
54| 1     FMA       fp12=fp6,fp0,fp11,fcr
53| 1     FMA       fp31=fp5,fp0,fp10,fcr
55| 1     LFL       fp5=a.rns9.(gr26,24)
52| 1     LFDU      fp6,gr26=a.rns9.(gr26,gr30,0)
55| 1     FMA       fp7=fp7,fp0,fp8,fcr
53| 1     LFL       fp8=a.rns9.(gr26,8)
54| 1     LFL       fp9=a.rns9.(gr26,16)
55| 1     LFL       fp10=a.rns9.(gr26,24)
52| 1     LFDU      fp0,gr26=a.rns9.(gr26,gr30,0)
52| 1     FMA       fp11=fp2,fp1,fp4,fcr
53| 1     LFL       fp1=a.rns9.(gr26,8)
54| 1     LFL       fp2=a.rns9.(gr26,16)
53| 1     FMA       fp31=fp31,fp4,fp13,fcr
54| 1     FMA       fp12=fp12,fp4,fp3,fcr
52| 1     LFL       fp13=b.rns8.(gr27,24)
55| 1     LFL       fp3=a.rns9.(gr26,24)
55| 1     FMA       fp4=fp7,fp4,fp5,fcr
52| 1     FMA       fp5=fp11,fp6,fp13,fcr
54| 1     FMA       fp6=fp12,fp13,fp9,fcr
53| 1     FMA       fp11=fp31,fp13,fp8,fcr
52| 1     LFDU      fp7,gr27=b.rns8.(gr27,32)
55| 1     FMA       fp8=fp4,fp13,fp10,fcr
 0| 0     BCT       ctr=CL.170,taken=100%(100,0)
```

Execution time is STILL dominated by LOAD instructions (20:16)
but the ratio is WAY better.

Matrix product 2: Loop unrolling

Why stop at $\times 4$? If it is such a good idea, apply it multiple times, in a 2D pattern

```
DO I=I4+1,IB,4
   T00 = C(I+0,J+0)
   T10 = C(I+1,J+0)
   T20 = C(I+2,J+0)
   T30 = C(I+3,J+0)
   T01 = C(I+0,J+1)
   T11 = C(I+1,J+1)
   T21 = C(I+2,J+1)
   T31 = C(I+3,J+1)
   A0 = A(I,1)
   DO K=1,KB
      B0 = B(K,J+0)
      B1 = B(K,J+1)
      T00 = T00 + A0*B0
      T01 = T01 + A0*B1
      A1 = A(I+1,K)
      A2 = A(I+2,K)
      T10 = T10 + A1*B0
      T11 = T11 + A1*B1
      A3 = A(I+3,K)
      T20 = T20 + A2*B0
      T21 = T21 + A2*B1
      T30 = T30 + A3*B0
      T31 = T31 + A3*B1
      A0 = A(I+0,K+1)
   END DO
   C(I+0,J+0) =  T00
   C(I+1,J+0) =  T10
   C(I+2,J+0) =  T20
   C(I+3,J+0) =  T30
   C(I+0,J+1) = T01
   C(I+1,J+1) = T11
   C(I+2,J+1) = T21
   C(I+3,J+1) = T31
END DO
```

What comes out of this?

Matrix product 2: Loop unrolling

```
103|                CL.313:
106| 000694  1     FMA       fp4=fp4,fp2,fp5,fcr
107| 000698  1     FMA       fp12=fp12,fp3,fp5,fcr
102| 00069C  1     LFL       fp31=b.rns8.(gr20,8)
102| 0006A0  1     LFDU      fp2,gr20=b.rns8.(gr20,16)
109| 0006A4  1     FMA       fp8=fp8,fp6,fp9,fcr
110| 0006A8  1     FMA       fp13=fp13,fp3,fp9,fcr
103| 0006AC  1     LFL       fp30=b.rns8.(gr22,8)
106| 0006B0  1     LFDU      fp25,gr23=a.rns9.(gr23,gr27,0)
111| 0006B4  1     FMA       fp11=fp11,fp6,fp10,fcr
112| 0006B8  1     FMA       fp29=fp29,fp3,fp10,fcr
109| 0006BC  1     LFL       fp26=a.rns9.(gr23,8)
111| 0006C0  1     LFL       fp28=a.rns9.(gr23,16)
113| 0006C4  1     LFDU      fp27,gr21=a.rns9.(gr21,gr27,0)
106| 0006C8  1     LFDU      fp5,gr23=a.rns9.(gr23,gr27,0)
  0| 0006CC  1     LRFL      fp6=fp2
102| 0006D0  1     FMA       fp0=fp0,fp7,fp31,fcr
103| 0006D4  1     FMA       fp1=fp1,fp7,fp30,fcr
109| 0006D8  1     LFL       fp9=a.rns9.(gr23,8)
113| 0006DC  1     LFDU      fp7,gr21=a.rns9.(gr21,gr27,0)
106| 0006E0  1     FMA       fp4=fp4,fp31,fp25,fcr
107| 0006E4  1     FMA       fp12=fp12,fp30,fp25,fcr
111| 0006E8  1     LFL       fp10=a.rns9.(gr23,16)
103| 0006EC  1     LFDU      fp3,gr22=b.rns8.(gr22,16)
109| 0006F0  1     FMA       fp8=fp8,fp31,fp26,fcr
110| 0006F4  1     FMA       fp13=fp13,fp30,fp26,fcr
111| 0006F8  1     FMA       fp11=fp11,fp31,fp28,fcr
112| 0006FC  1     FMA       fp29=fp29,fp30,fp28,fcr
102| 000700  1     FMA       fp0=fp0,fp27,fp2,fcr
103| 000704  1     FMA       fp1=fp1,fp27,fp3,fcr
  0| 000708  0     BCT       ctr=CL.313,taken=100%(100,0)
```

Ok, now the ratio is 13:16, much better...

Matrix product: Unrolling 4x4

```
DO K=1,KB
   B0 = B(K,J+0)
   B1 = B(K,J+1)
   B2 = B(K,J+2)
   B3 = B(K,J+3)
   T00 = T00 + A0*B0
   T01 = T01 + A0*B1
   T02 = T02 + A0*B2
   T03 = T03 + A0*B3
   A1 = A(I+1,K)
   A2 = A(I+2,K)
   T10 = T10 + A1*B0
   T11 = T11 + A1*B1
   T12 = T12 + A1*B2
   T13 = T13 + A1*B3
   A3 = A(I+3,K)
   T20 = T20 + A2*B0
   T21 = T21 + A2*B1
   T22 = T22 + A2*B2
   T23 = T23 + A2*B3
   T30 = T30 + A3*B0
   T31 = T31 + A3*B1
   T32 = T32 + A3*B2
   T33 = T33 + A3*B3
   A0 = A(I+0,K+1)
END DO
```

```
  0|          CL.302:
116| 0006F8 1     FMA       fp4=fp4,fp28,fp22,fcr
  0| 0006FC 1     LRFL      fp21=fp26
115| 000700 1     LFDU      fp20,gr25=b.rns8.(gr25,8)
117| 000704 1     FMA       fp8=fp8,fp28,fp26,fcr
118| 000708 1     FMA       fp12=fp12,fp28,fp23,fcr
134| 00070C 1     LFDU      fp28,gr20=a.rns9.(gr20,gr31,0)
129| 000710 1     FMA       fp30=fp30,fp23,fp25,fcr
133| 000714 1     FMA       fp31=fp31,fp23,fp24,fcr
124| 000718 1     FMA       fp13=fp13,fp23,fp27,fcr
130| 00071C 1     FMA       fp3=fp3,fp29,fp24,fcr
131| 000720 1     FMA       fp7=fp7,fp22,fp24,fcr
115| 000724 1     FMA       fp0=fp0,fp28,fp20,fcr
126| 000728 1     FMA       fp2=fp2,fp29,fp25,fcr
132| 00072C 1     FMA       fp11=fp11,fp26,fp24,fcr
127| 000730 1     FMA       fp6=fp6,fp22,fp25,fcr
128| 000734 1     FMA       fp10=fp10,fp26,fp25,fcr
121| 000738 1     FMA       fp1=fp1,fp29,fp27,fcr
122| 00073C 1     FMA       fp5=fp5,fp22,fp27,fcr
117| 000740 1     LFDU      fp26,gr23=b.rns8.(gr23,8)
116| 000744 1     LFDU      fp22,gr22=b.rns8.(gr22,8)
123| 000748 1     FMA       fp9=fp9,fp21,fp27,fcr
121| 00074C 1     LFDU      fp27,gr24=a.rns9.(gr24,gr31,0)
118| 000750 1     LFDU      fp23,gr21=b.rns8.(gr21,8)
126| 000754 1     LFL       fp25=a.rns9.(gr24,8)
130| 000758 1     LFL       fp24=a.rns9.(gr24,16)
  0| 00075C 1     LRFL      fp29=fp20
  0| 000760 0     BCT       ctr=CL.302,taken=100%(100,0)
```

Ratio: (10:16). So, FLOP dominate. Does it work in practice?

Measurements, take 2

Performance MFLOPS on IBM POWER5 (peak 7600 MFLOPS)										
Alg.	100	200	300	400	500	600	700	800	900	1000
ijk	1519	1554	509	351	271	263	251	247	207	131
kij	885	330	264	218	178	67	63	55	54	57
4x1	2451	2254	1366	982	693	671	639	688	575	448
4x4	4385	3883	3155	2636	1785	2030	1671	2081	1505	1371

This is good for *pipeline and internal parallelism* but it is not using well the *memory hierarchy*.

Matrix product

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad c_{ij} = c_{ij} + \sum_{k=1,n} a_{ik} b_{kj} \qquad i, j = 1 \ldots n$$

Matrix product

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad c_{ij} = c_{ij} + \sum_{k=1,n} a_{ik} b_{kj} \qquad i, j = 1 \ldots n$$

In the above we only ever use associativity, not commutativity

Matrix product

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad c_{ij} = c_{ij} + \sum_{k=1,n} a_{ik} b_{kj} \qquad i,j = 1 \ldots n$$

In the above we only ever use associativity, not commutativity
Therefore it is possible to rewrite as:

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad C_{ij} = C_{ij} + \sum_{k=1,n/q} A_{ik} B_{kj} \qquad i,j = 1 \ldots n/q$$

At each step we are accessing A_{ik} and B_{kj}, two $q \times q$ matrices, therefore $2q^2$ data items while doing $2q^3$ floating-point operations; thus

Matrix product

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad c_{ij} = c_{ij} + \sum_{k=1,n} a_{ik} b_{kj} \qquad i,j = 1 \ldots n$$

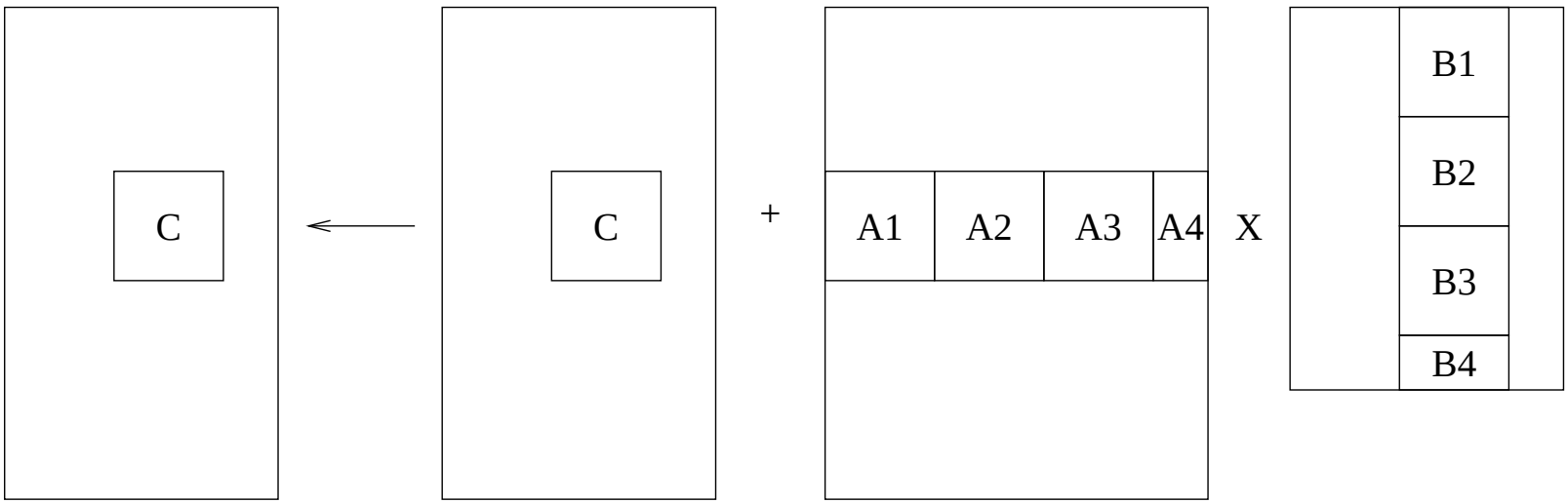
In the above we only ever use associativity, not commutativity
Therefore it is possible to rewrite as:

$$C \leftarrow C + A \cdot B \qquad \Leftrightarrow \qquad C_{ij} = C_{ij} + \sum_{k=1,n/q} A_{ik} B_{kj} \qquad i,j = 1 \ldots n/q$$

At each step we are accessing A_{ik} and B_{kj}, two $q \times q$ matrices, therefore $2q^2$ data items while doing $2q^3$ floating-point operations; thus

Each data item is reused q times.

Matrix product: blocking



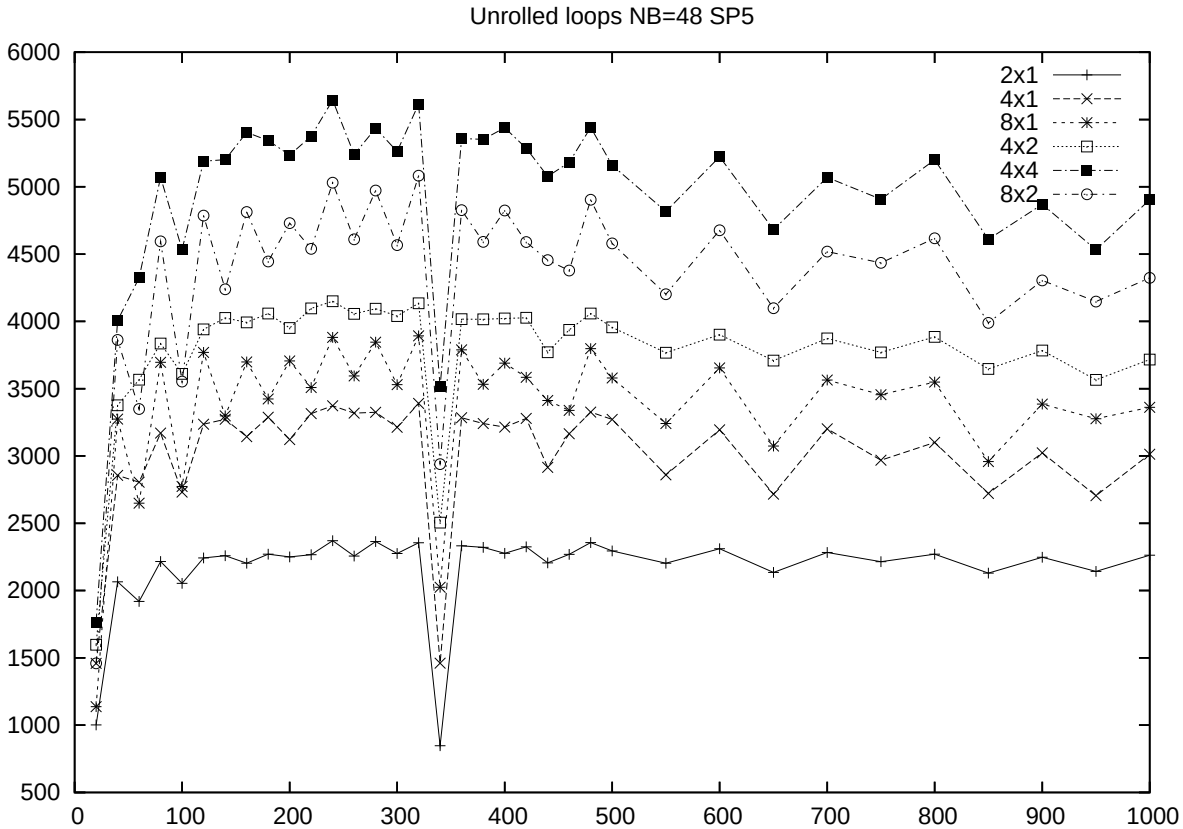
Partitioning of large matrix−matrix products

This is also the basis of parallelization; variants of the loop nests must be explored at the block level.

Matrix product: blocking

```
for ii=1:nb:m
  ib = min(nb,m-ii+1)
  for jj=1:nb:n
    jb = min(nb,n-jj+1)
    for kk=1:nb:k
      kb = min(nb,k-kk+1)
      #
      #      .....compute product on blocks......
      #
      c(ii:ii+ib-1,jj:jj+jb-1) = c(ii:ii+ib-1,jj:jj+jb-1) +
                a(ii:ii+ib-1,kk:kk+kb-1) * b(kk:kk+kb-1,jj:jj+jb-1) ;
    end
  end
end
```

Measurements take 3: (Power5)



ATLAS Project: code generator exploring automatically all (most) possible code variants, and choosing the best.
`http://math-atlas.sourceforge.net/`