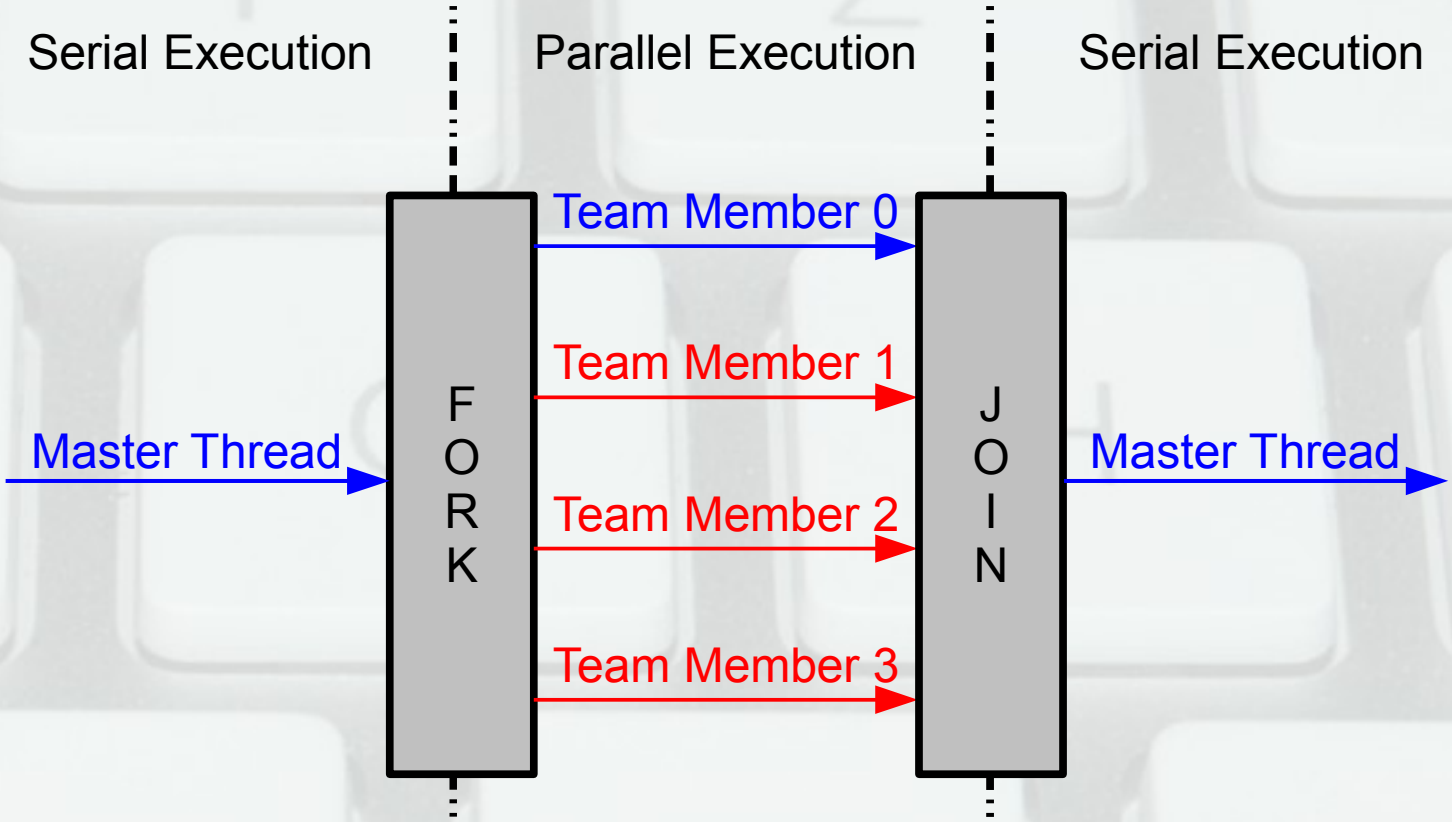
A Brief Introduction to

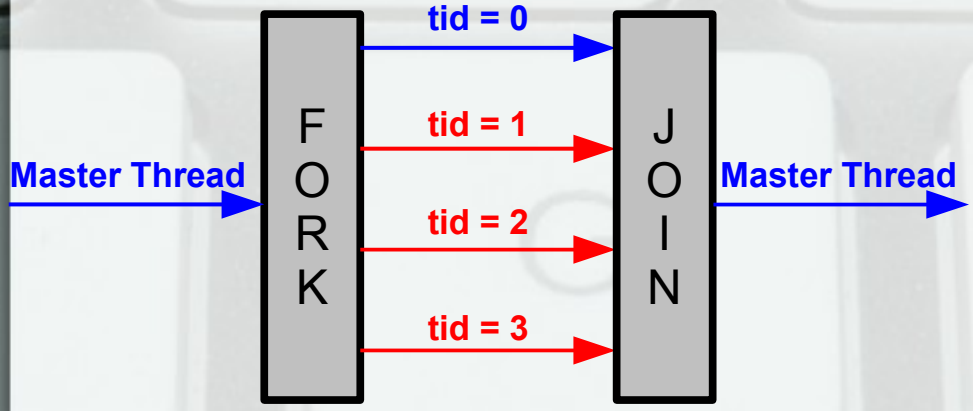
BASIC IDEA



Note

# of threads = # of cores

HELLO WORLD

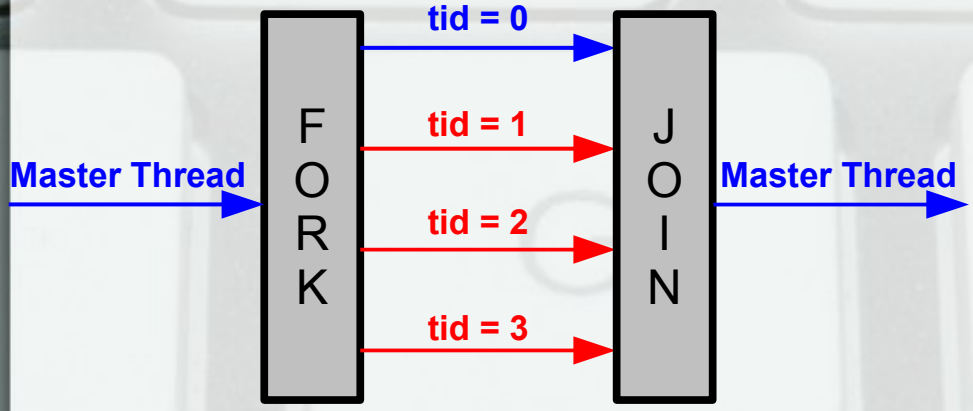


The master thread creates a *team* of threads.

Example 1 – Simple ID reporting

```
#include <omp.h>

int main()
{
     int tid, num;

     // Parallel
#pragma omp parallel private(tid)
{    // Fork Here

     // All Threads
     tid = omp_get_thread_num();
     printf("My thread id: %i\n", tid);

     // Only Master Thread
     if (tid == 0)
     {
          num = omp_get_num_threads();
          printf("Num threads: %i\n", num);
     }

}    // Join Here
}
```

HELLO WORLD



The master thread creates a *team* of threads.

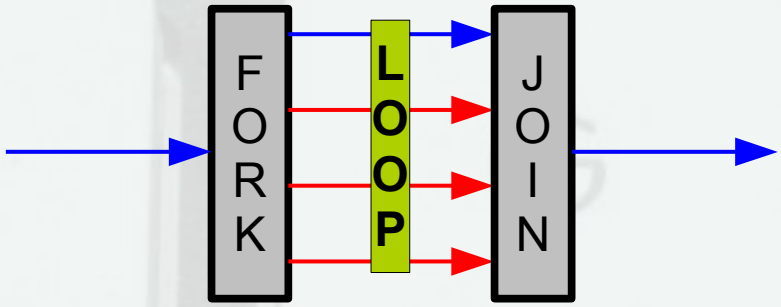
Example 1 – Simple ID reporting

```
#include <omp.h>

int main()
{
     int tid, num;

     // Parallel
#pragma omp parallel private(tid)
{    // Fork Here

     // All Threads
     tid = omp_get_thread_num();
     printf("My thread id: %i\n", tid);

     // Only Master Thread
     if (tid == 0)
     {
          num = omp_get_num_threads();
          printf("Num threads: %i\n", num);
     }

}    // Join Here
}
```

Every thread gets its own copy of `tid`.

2
Types of Parallelism

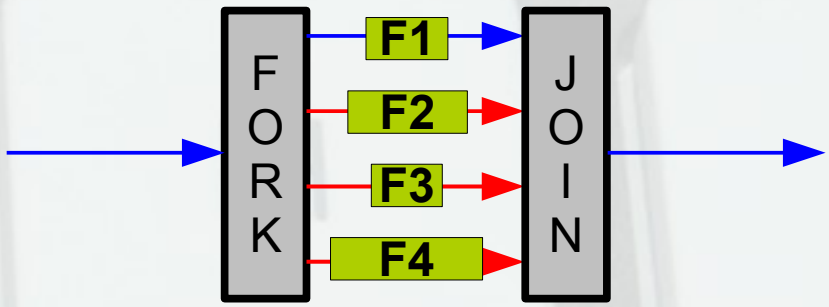
DATA PARALLELISM



Cores execute ***same instructions***

...but operate on ***different data.***

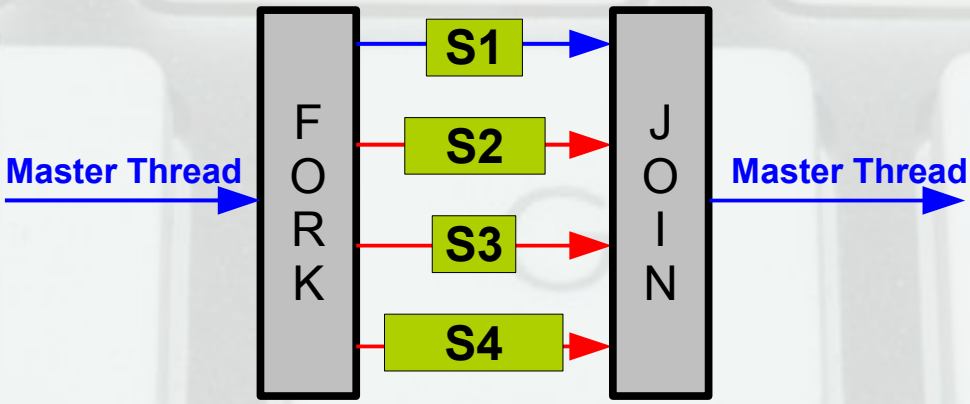
FUNCTIONAL PARALLELISM



Cores execute ***different instructions***

...and *can* read same data &
should write different data.

FUNCTIONAL PARALLELISM

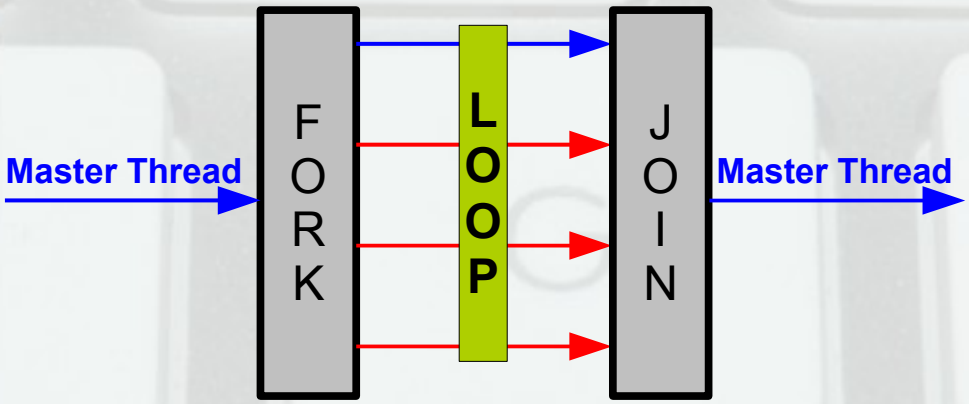


The *team* works in parallel
to process **different sections**

Example 2 – Functional Parallelism

```
#include <omp.h>
#define N 50

int main()
{
     int i;
     float a[N], b[N], c[N], d[N];

     ...

     // Parallel Function
#pragma omp parallel for \
     shared(a,b,c,d) privite(i)
     {
          #pragma omp sections
          {
               #pragma omp section
               for (i=0; i<N; i++)
                    c[i] = a[i] + b[i];

               #pragma omp section
               for (i=0; i<N; i++)
                    d[i] = a[i] * b[i];
          }
     }
     // Serial Code
     ...
```

DATA PARALLELISM



The *team* works in parallel to process the `for loop`

Example 3 – Data Parallelism

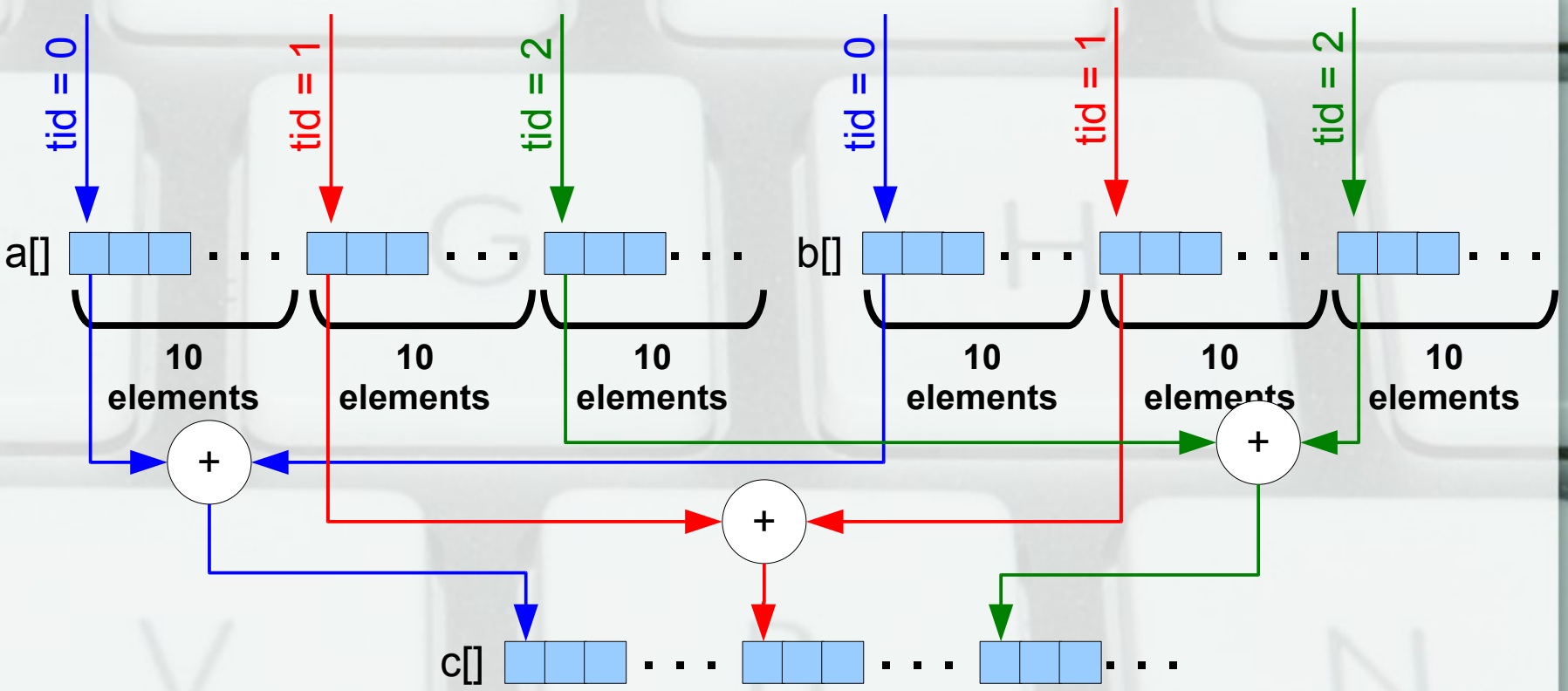
```
#include <omp.h>

int main()
{
     int i;
     float a[50], b[50], c[50];

     // Serial Initialization
     for (i=0; i<50; i++)
          a[i] = b[i] = 2 * i;

     // Parallel Addition
#pragma omp parallel for \
     shared(a,b,c) private(i)
     for (i=0; i<n; i++)
          c[i] = a[i] + b[i]

     // Serial Code
     ...
```

...how does this work?

DATA PARALLELISM

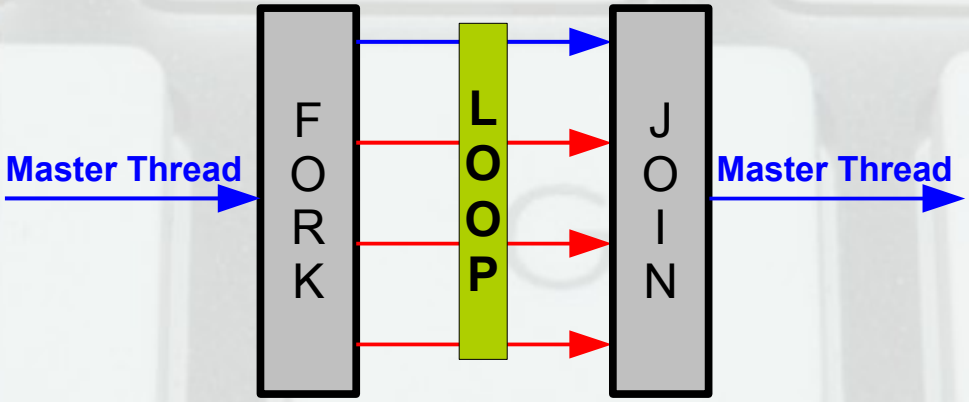
Threads are assigned *chunks* of work.
Here, **CHUNK_SIZE = 10**

When a thread **finishes** a chunk, it will be assigned another.



DATA PARALLELISM

CHUNK_SIZE can be manually set, or you can use the (implementation dependent) default.



The *team* works in parallel to process the `for loop`

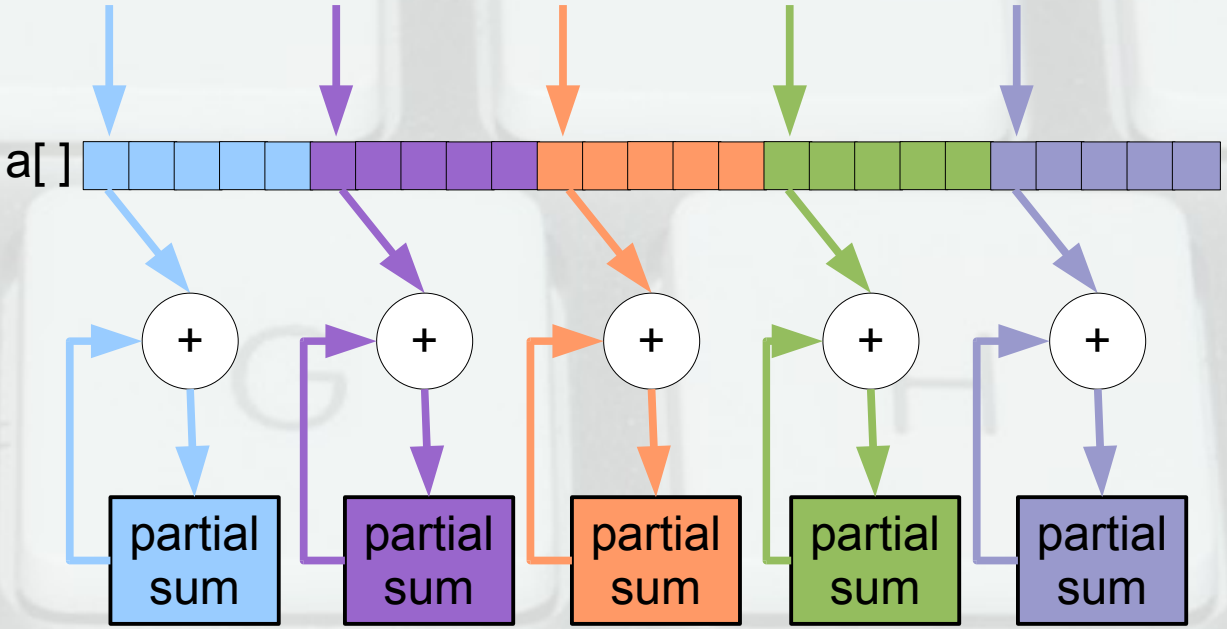
Example 3 – Data Parallelism

```
#include <omp.h>

int main()
{
     int i;
     int CHUNK_SIZE = 10;
     float a[50], b[50], c[50];

     // Serial Initialization
     for (i=0; i<50; i++)
          a[i] = b[i] = 2 * i;

     // Parallel Addition
#pragma omp parallel for \
     shared(a,b,c) private(i) \
     schedule(static, CHUNK_SIZE)
     for (i=0; i<n; i++)
          c[i] = a[i] + b[i]

     // Serial Code
     ...
```

PRACTICAL APPLICATIONS

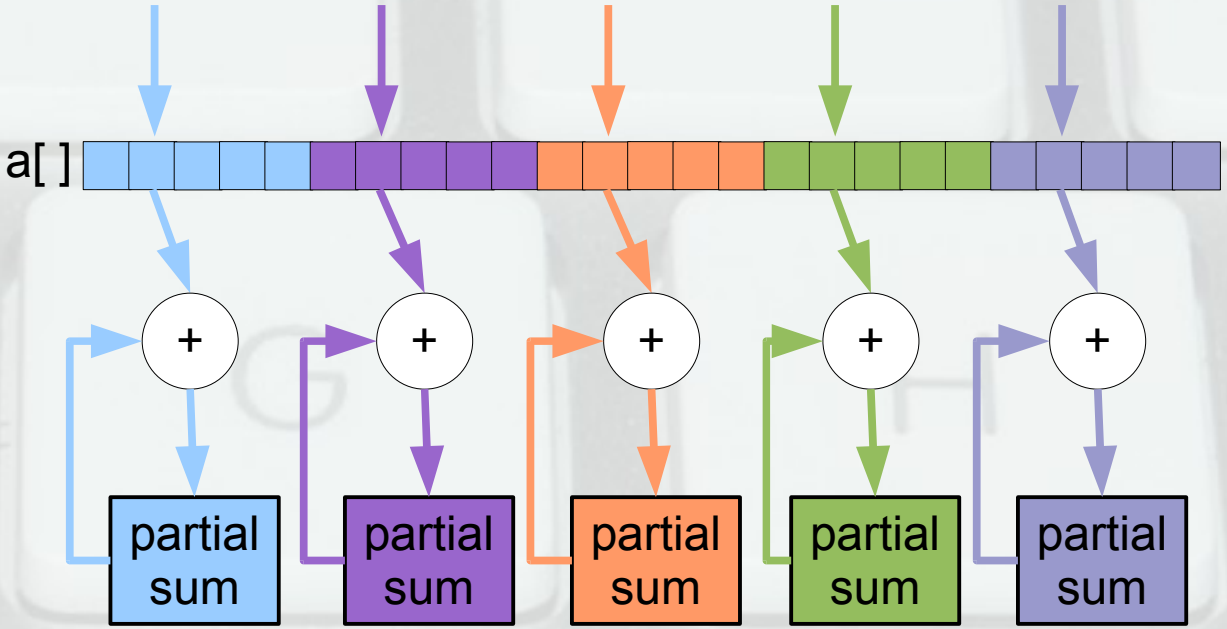
USEFUL TOOL:

PARALLEL SUM REDUCTION



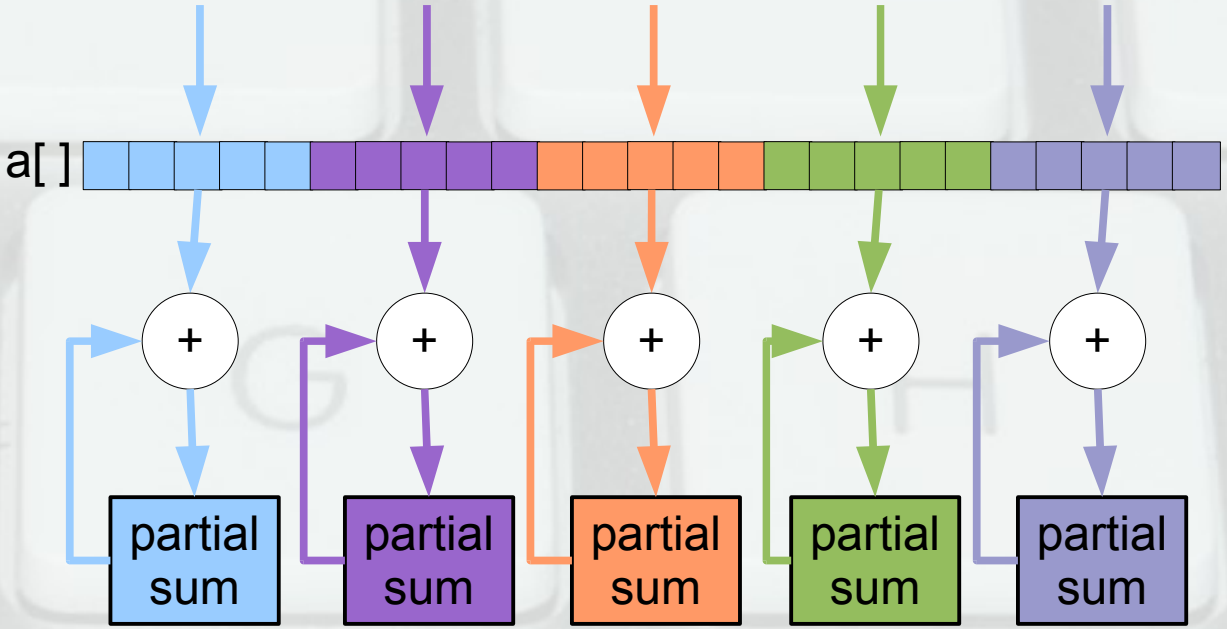
USEFUL TOOL:

PARALLEL SUM REDUCTION



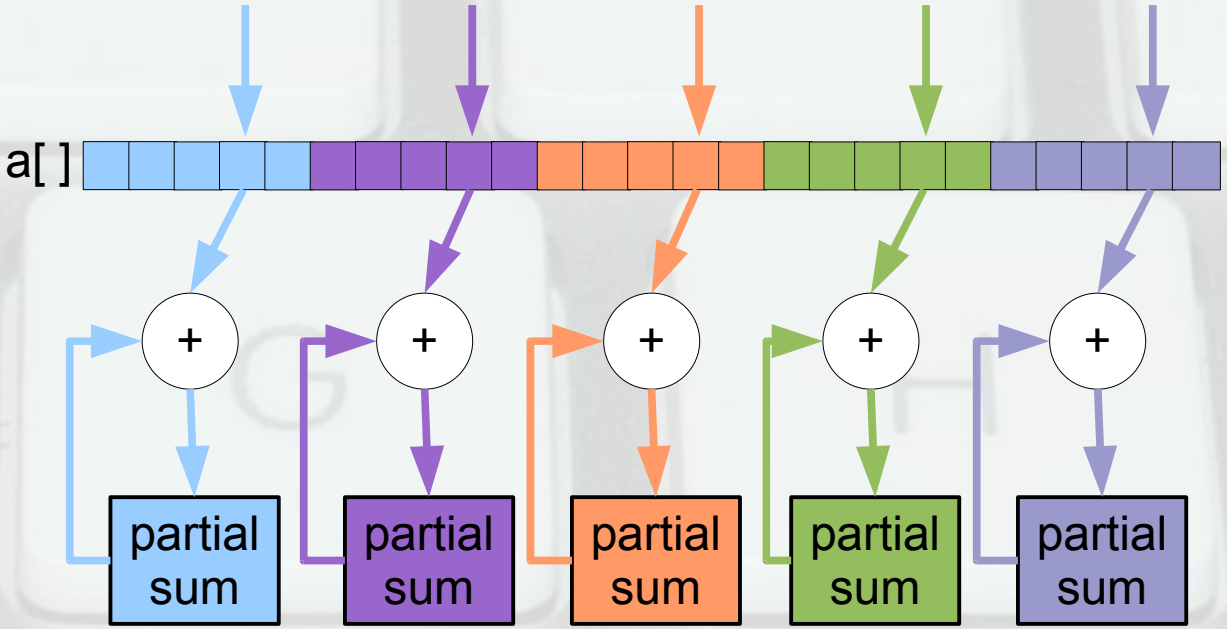
USEFUL TOOL:

PARALLEL SUM REDUCTION



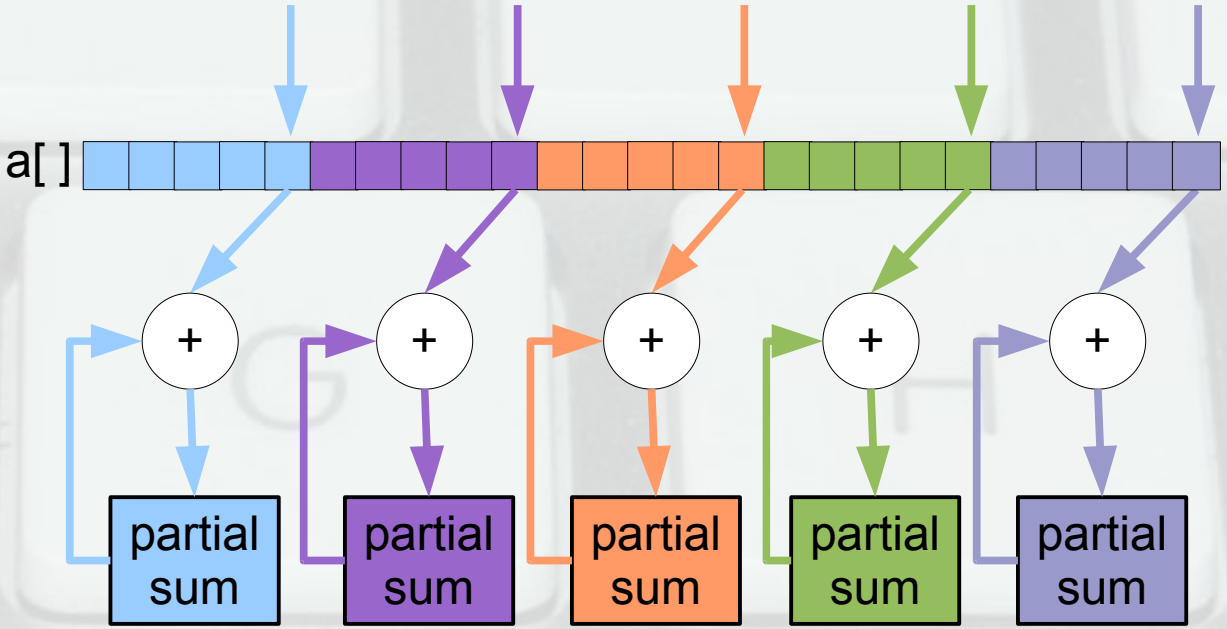
USEFUL TOOL:

PARALLEL SUM REDUCTION



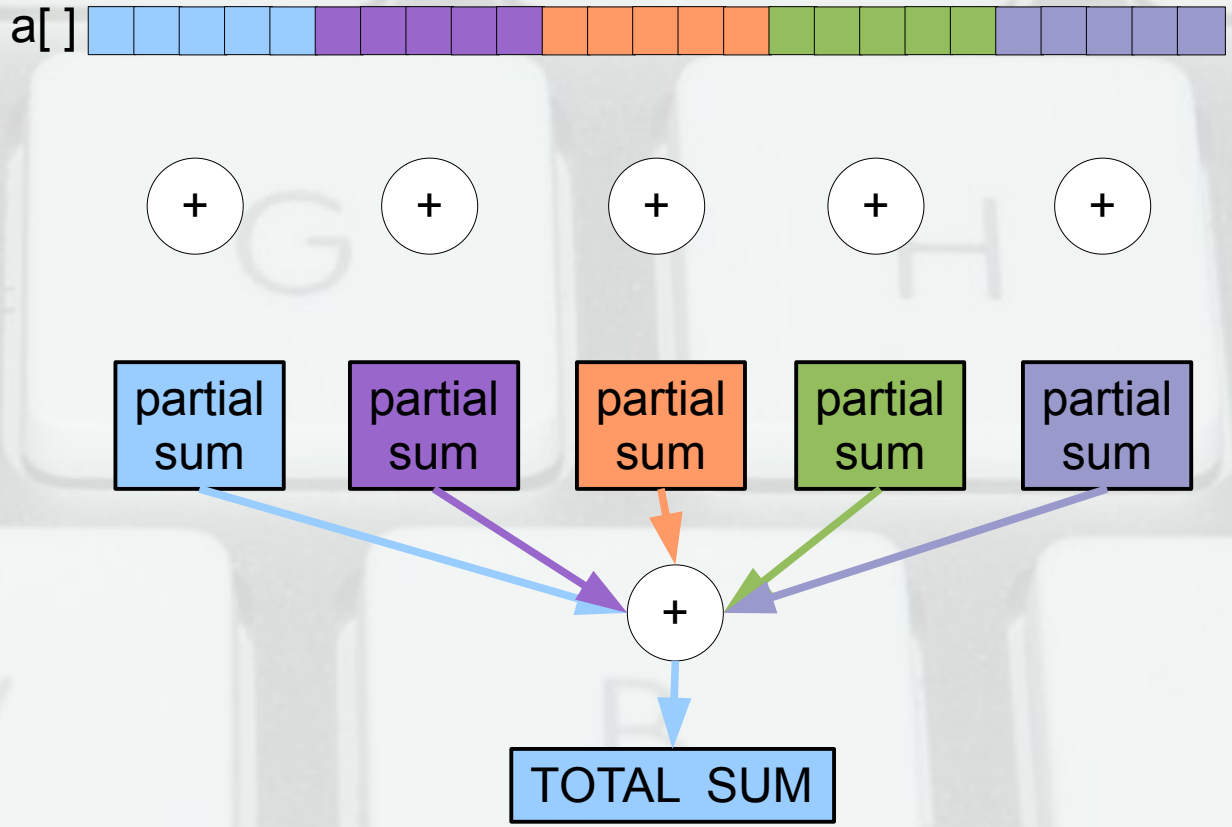
USEFUL TOOL:

PARALLEL SUM REDUCTION



USEFUL TOOL:

PARALLEL SUM REDUCTION



OpenMP has sum reduction built in!

Example 4 – Sum Reduction

```
#include <omp.h>
#include <stdio.h>

int main()
{
    int i, N;
    float result;
    float* a = (float*)malloc(...);
    . . .

    // Parallel Sum Reduction
    #pragma omp parallel for \
        private(i) \
        reduction(+:result)
    for (i=0; i<N; i++)
        result += a[i];

    free(a);

    printf("Total Sum: %f\n", result);

    return(0);
}
```

OpenMP has sum reduction built in!

Example 4 – Sum Reduction

```
#include <omp.h>
. . .

int main()
{
     int i, N;
     float result;
     float* a = (float*)malloc(...);
     . . .

     // Parallel Sum Reduction
     #pragma omp parallel for \
          private(i) \
          reduction(+:result)
     for (i=0; i<N; i++)
          result += a[i];

     free(a);

     printf("Total Sum: %f\n", result);

     return(0);
}
```

Partial Sums.

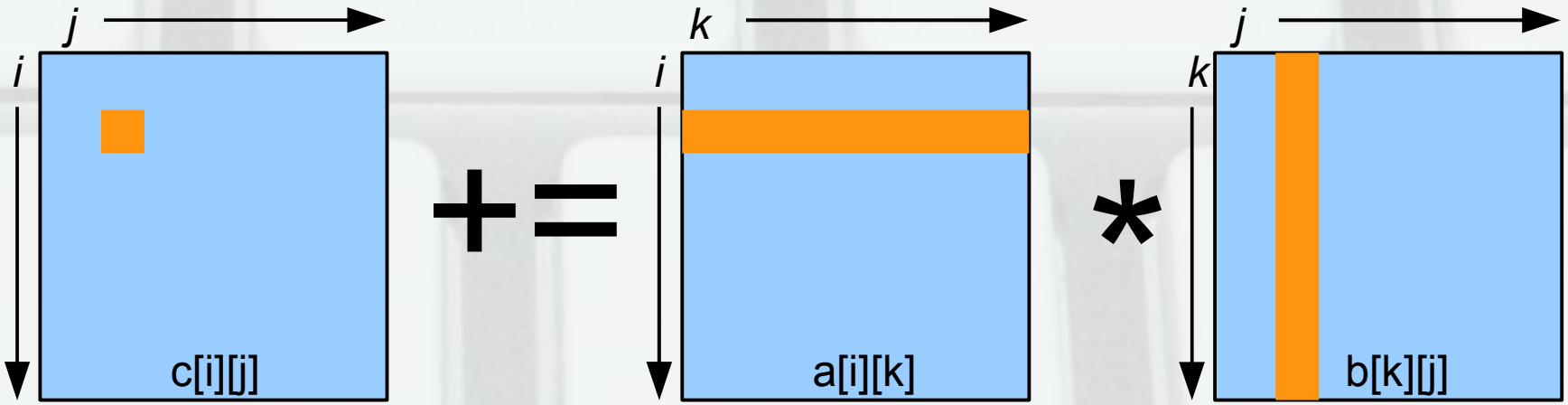
`result` is private between threads until the loop is complete. Then all priviate copies are summed.

Total Sum.

MATRIX MULTIPLY

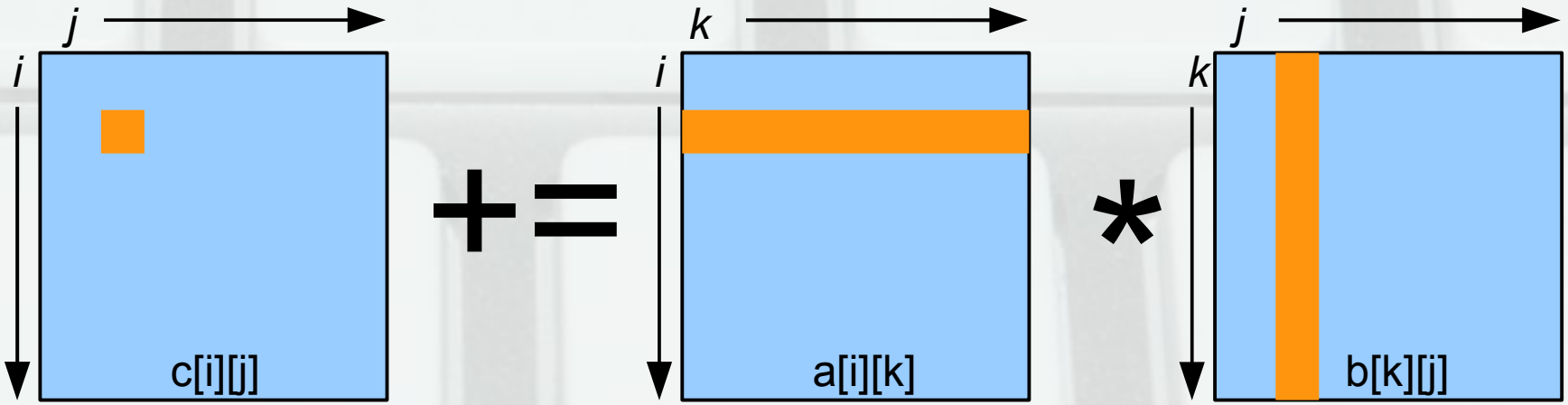
Example 4 – Matrix Multiply (Serial)

```
#include <omp.h>
. . .

int main()
{
     int i,j,k;
     float a[10][10], b[10][10], c[10][10];

     for (i=0; i<10; i++)
          for (j=0; j<10; j++)
               for (k=0; k<10; k++)
                    c[i][j] += a[i][k] * b[k][j];

     return 0;
}
```



Example 4 – Matrix Multiply (OpenMP)

```
#include <omp.h>

int main()
{
     int i,j,k;
     float a[10][10], b[10][10], c[10][10];

     #pragma omp parallel for private(i,j,k)
     for (i=0; i<10; i++)
          for (j=0; j<10; j++)
               for (k=0; k<10; k++)
                    c[i][j] += a[i][k] * b[k][j];

     return 0;
}
```



HOW TO COMPILE OpenMP CODE

```
user@host:~/code$ ls
mycode.c

user@host:~/code$ gcc -fopenmp -o mycode mycode.c

user@host:~/code$ ls
mycode.c mycode

user@host:~/code$ time ./mycode
```

More Useful Information:

- http://openmp.org
- https://computing.llnl.gov/tutorials/openMP/
- http://software.intel.com/en-us/articles/getting-started-with-openmp/
- http://openmp.org/mp-documents/omp-hands-on-SC08.pdf